

SEFM 2010 School

Advanced applications of model-checking techniques

Applications in human-computer interaction

Qualitative and Quantitative Formal Analysis in HCI

Part 1: Qualitative Analysis

Antonio Cerone

United Nations University

International Institute for Software Technology

Macau SAR China

email: antonio@iist.unu.edu

web: www.iist.unu.edu

Contents

1. Formal Tools and ATM Example
2. HCI Concepts
3. Modelling Human Behaviour
 - (a) ATM Example Revisited
 - (b) Cognitive Errors
 - (c) Attention
 - (d) History of Formal HCI
4. Task Failures and ATC Case Study
5. References

Formal Tools

Process Algebras

- event-based formal specification languages

Process Algebras

- event-based formal specification languages
- basic entities: **actions** and **processes**

Process Algebras

- event-based formal specification languages
- basic entities: **actions** and **processes**
- **processes**

Process Algebras

- event-based formal specification languages
- basic entities: **actions** and **processes**
- **processes**
 - evolve **by** performing **actions**

Process Algebras

- event-based formal specification languages
- basic entities: **actions** and **processes**
- **processes**
 - evolve **by** performing **actions**
 - **are** composed using **operators**

Examples of Process Algebras

- CSP — Communicating Sequential Processes
- CCS — Calculus of Communicating Systems
- CirCal — Circuit Calculus
- Lotos — Language of Temporal Ordering Specs
- ACP — Algebra of Communicating Processes

Examples of Process Algebras

- CSP** — Communicating Sequential Processes
- CCS — Calculus of Communicating Systems
- CirCal — Circuit Calculus
- Lotos — Language of Temporal Ordering Specs
- ACP — Algebra of Communicating Processes

Concurrency Workbench

The Concurrency Workbench of New Century
(CWB-NC) supports

Concurrency Workbench

The Concurrency Workbench of New Century (CWB-NC) supports

- modelling using several process algebras: CCS and some extensions, Lotos, CSP
- simulation
- model-checking

Starting CWB-NC

```
shell>
```

Starting CWB-NC

```
shell> cwb-nc csp
```

```
The Concurrency Workbench of the New Ce  
(Version 1.2 --- June, 2000)
```

```
cwn-nc>
```

Starting CWB-NC

```
shell> cwb-nc csp
```

```
The Concurrency Workbench of the New Ce  
(Version 1.2 --- June, 2000)
```

```
cwn-nc> load atm-machine.csp  
Execution time (user,system,gc,real):(0  
cwn-nc>
```

Starting CWB-NC

```
shell> cwb-nc csp
```

The Concurrency Workbench of the New Century
(Version 1.2 --- June, 2000)

```
cwn-nc> load atm-machine.csp
```

```
Execution time (user,system,gc,real):(0
```

```
cwn-nc> help
```

Available CWB-NC commands are:

 caching {on | off}

 cat identifier

 cd directory

 ...

CSP Notation

Communication Sequential Processes

Actions: a, b, c, \dots

Processes: P, Q, R, \dots

CSP Notation

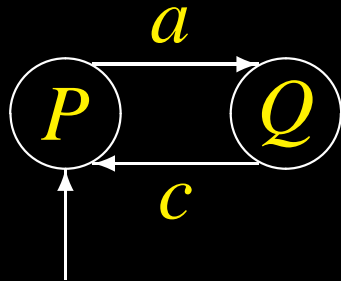
Communication Sequential Processes

Actions: a, b, c, \dots

Processes: P, Q, R, \dots

Basic Operators

Prefix: \rightarrow $P = a \rightarrow Q, \quad Q = c \rightarrow P$



CSP Notation

Communication Sequential Processes

Actions: a, b, c, \dots

Processes: P, Q, R, \dots

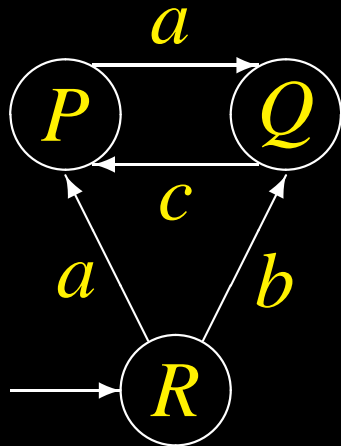
Basic Operators

Prefix: \rightarrow

$$P = a \rightarrow Q, \quad Q = c \rightarrow P$$

Choice: \square

$$R = (a \rightarrow P) \square (b \rightarrow Q)$$



CSP Notation

Communication Sequential Processes

Actions: a, b, c, \dots

Processes: P, Q, R, \dots

Basic Operators

Prefix: \rightarrow

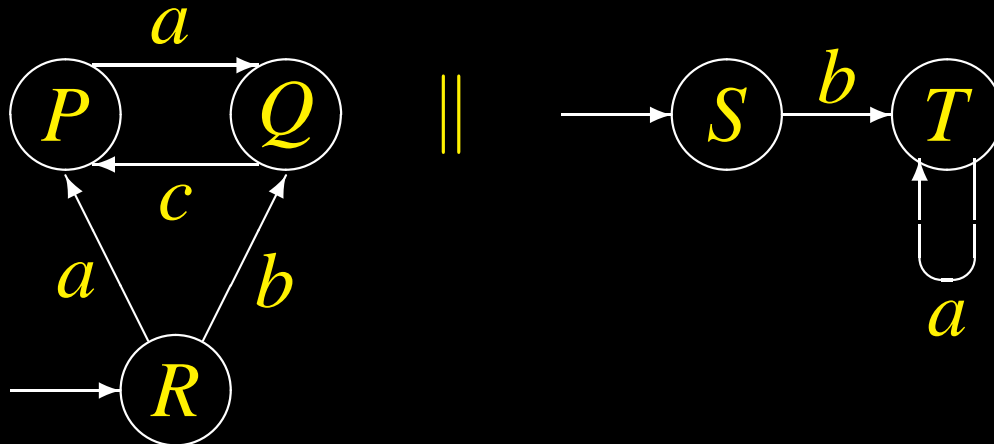
$$P = a \rightarrow Q, \quad Q = c \rightarrow P$$

Choice: \square

$$R = (a \rightarrow P) \square (b \rightarrow Q)$$

Parallel: \parallel

$$R \parallel S, \quad S = b \rightarrow T, \quad T = a \rightarrow T$$



CSP Notation

Communication Sequential Processes

Actions: a, b, c, \dots

Processes: P, Q, R, \dots

Basic Operators

Prefix: \rightarrow

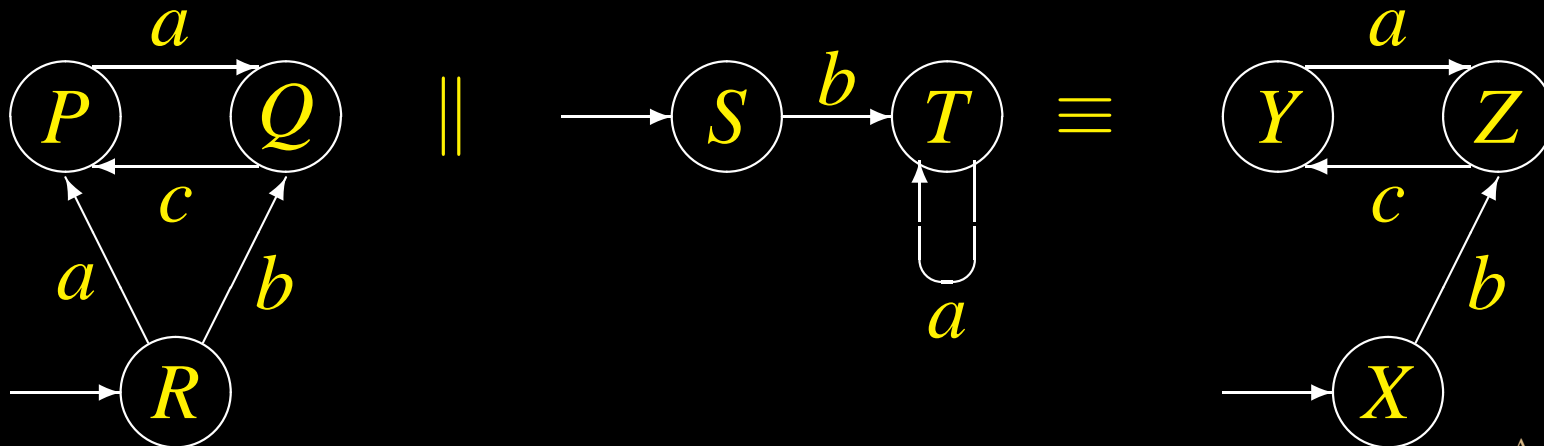
$$P = a \rightarrow Q, \quad Q = c \rightarrow P$$

Choice: \square

$$R = (a \rightarrow P) \square (b \rightarrow Q)$$

Parallel: \parallel

$$R \parallel S, \quad S = b \rightarrow T, \quad T = a \rightarrow T$$



CSP Notation

Communication Sequential Processes

Actions: a, b, c, \dots

Processes: P, Q, R, \dots

Basic Operators

Prefix: \rightarrow

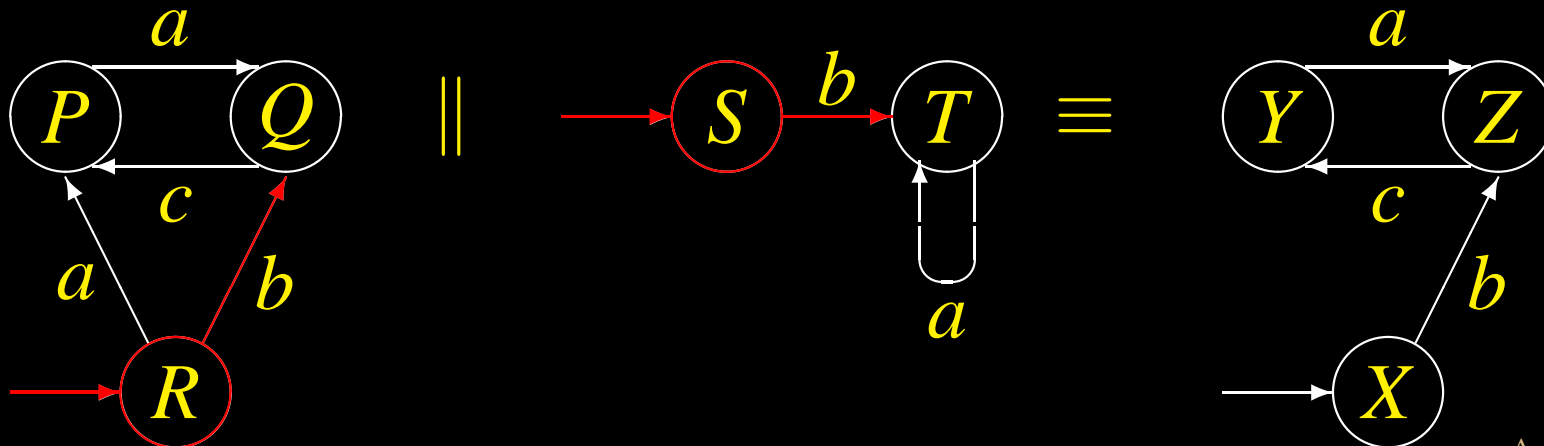
$$P = a \rightarrow Q, \quad Q = c \rightarrow P$$

Choice: \square

$$R = (a \rightarrow P) \square (b \rightarrow Q)$$

Parallel: \parallel

$$R \parallel S, \quad S = b \rightarrow T, \quad T = a \rightarrow T$$



CSP Notation

Communication Sequential Processes

Actions: a, b, c, \dots

Processes: P, Q, R, \dots

Basic Operators

Prefix: \rightarrow

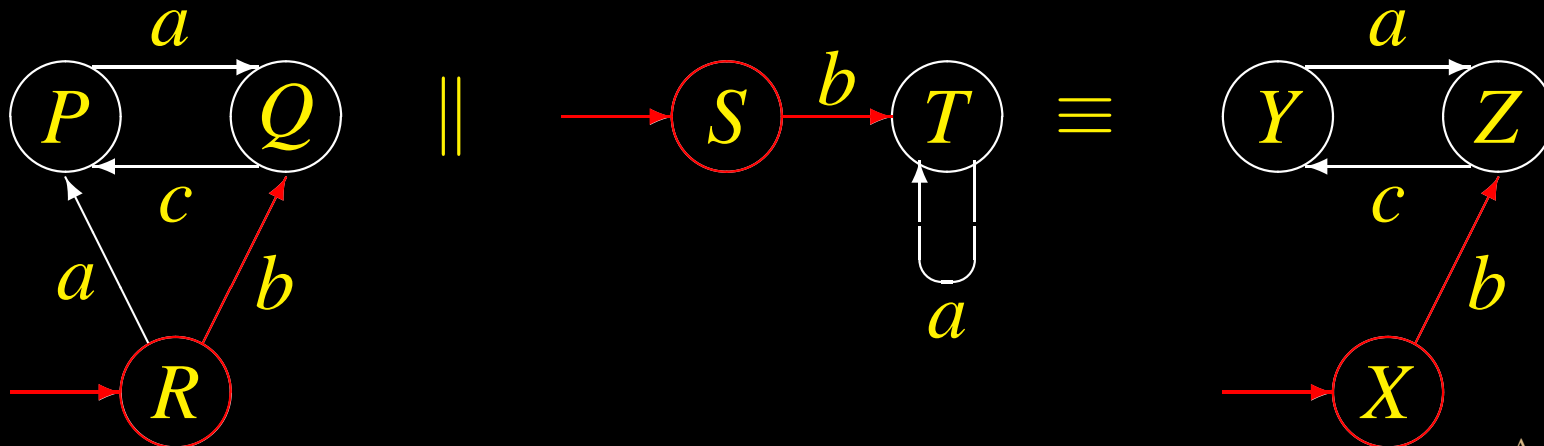
$$P = a \rightarrow Q, \quad Q = c \rightarrow P$$

Choice: \square

$$R = (a \rightarrow P) \square (b \rightarrow Q)$$

Parallel: \parallel

$$R \parallel S, \quad S = b \rightarrow T, \quad T = a \rightarrow T$$



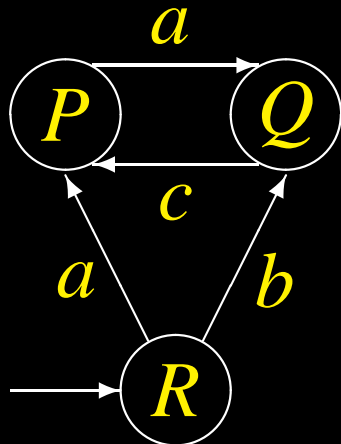
CWB-NC Syntax: R and S

$$P = a \rightarrow Q, \quad Q = c \rightarrow P, \quad R = (a \rightarrow P) \parallel (b \rightarrow Q)$$

```
proc P = a -> Q
```

```
proc Q = c -> P
```

```
proc R = a -> P [ ] b -> Q
```



CWB-NC Syntax: R and S

$$P = a \rightarrow Q, \quad Q = c \rightarrow P, \quad R = (a \rightarrow P) \parallel (b \rightarrow Q)$$

```
proc P = a -> Q
```

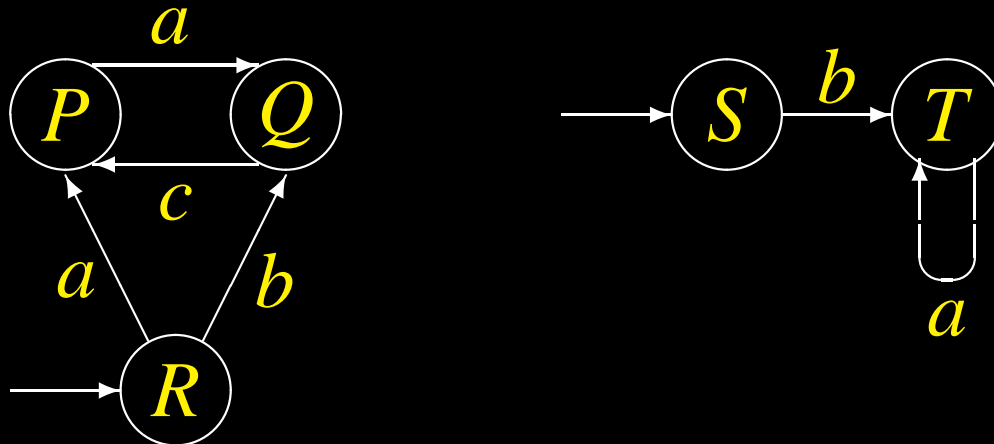
```
proc Q = c -> P
```

```
proc R = a -> P [ ] b -> Q
```

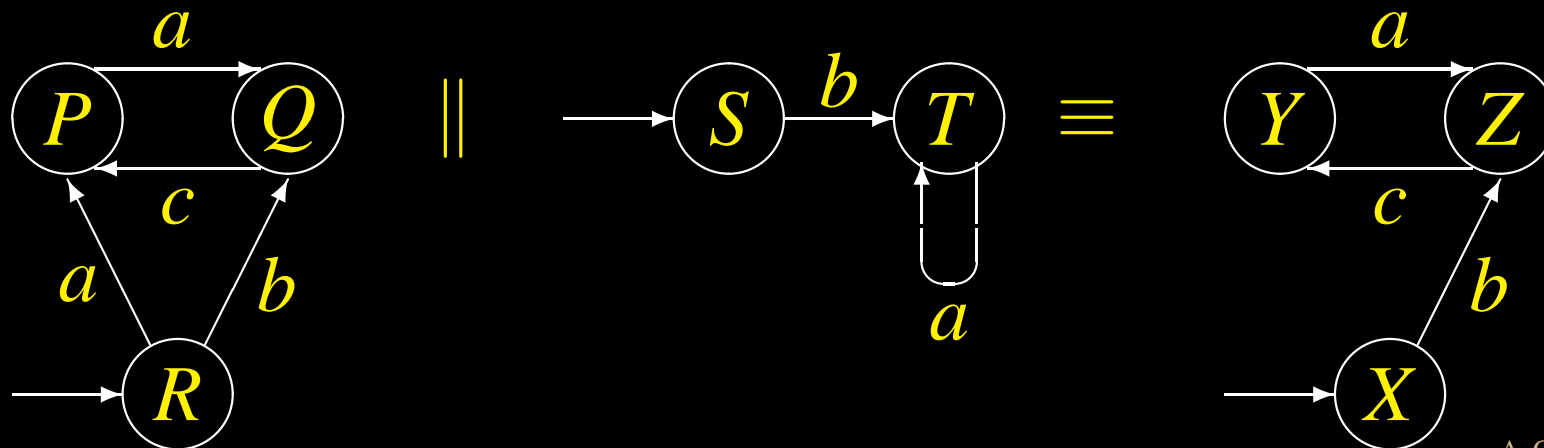
$$S = b \rightarrow T, \quad T = a \rightarrow T$$

```
proc S = b -> T
```

```
proc T = a -> T
```

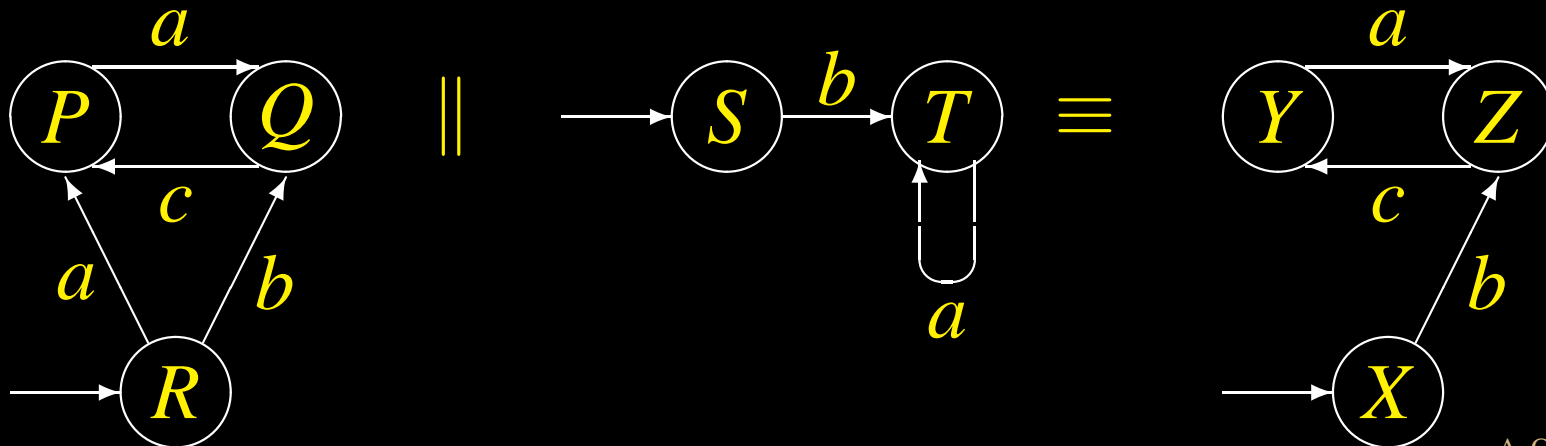


CSP: Synchronisation Set



CSP: Synchronisation Set

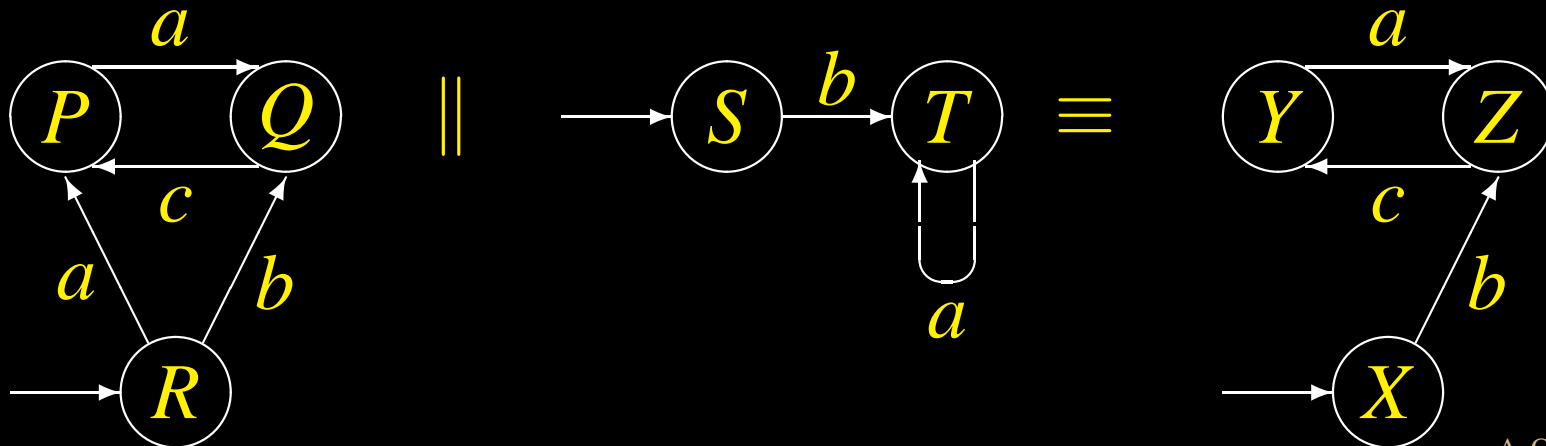
$$X = R \parallel S \Longrightarrow X = R[\parallel \{a, b\} \parallel] S$$



CSP: Synchronisation Set

$$X = R \parallel S \Longrightarrow X = R[\parallel \{a, b\} \parallel] S$$

$$\text{sync_ab} = \{a, b\} \Longrightarrow X = R[\parallel \text{sync_ab} \parallel] S$$



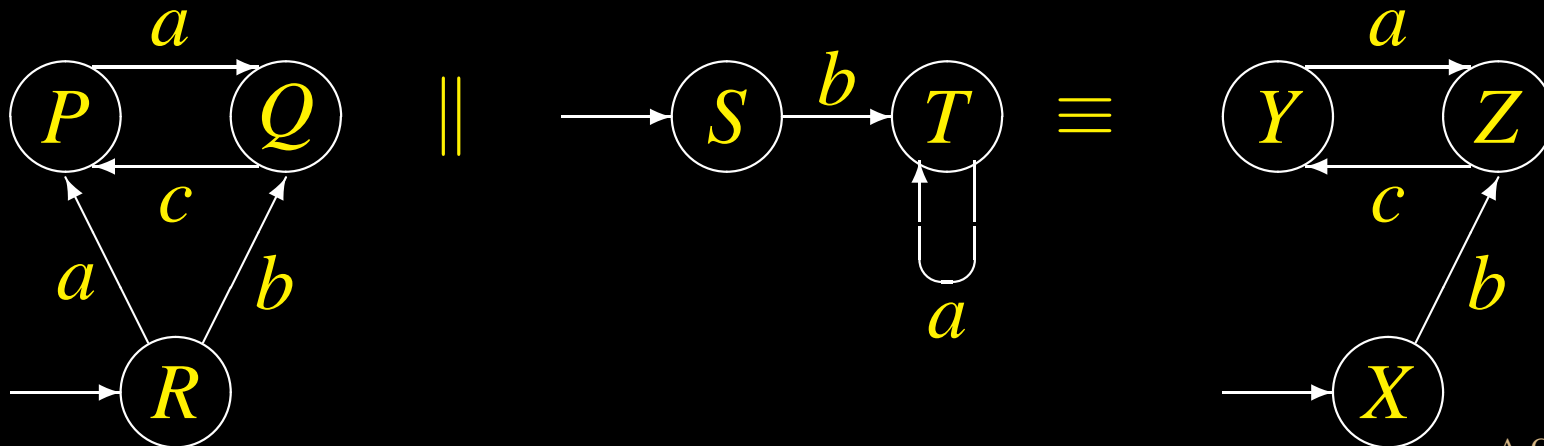
CSP: Synchronisation Set

$$X = R \parallel S \implies X = R[\parallel \{a, b\} \parallel] S$$

$$\text{sync_ab} = \{a, b\} \implies X = R[\parallel \text{sync_ab} \parallel] S$$

set sync-ab = { a, b }

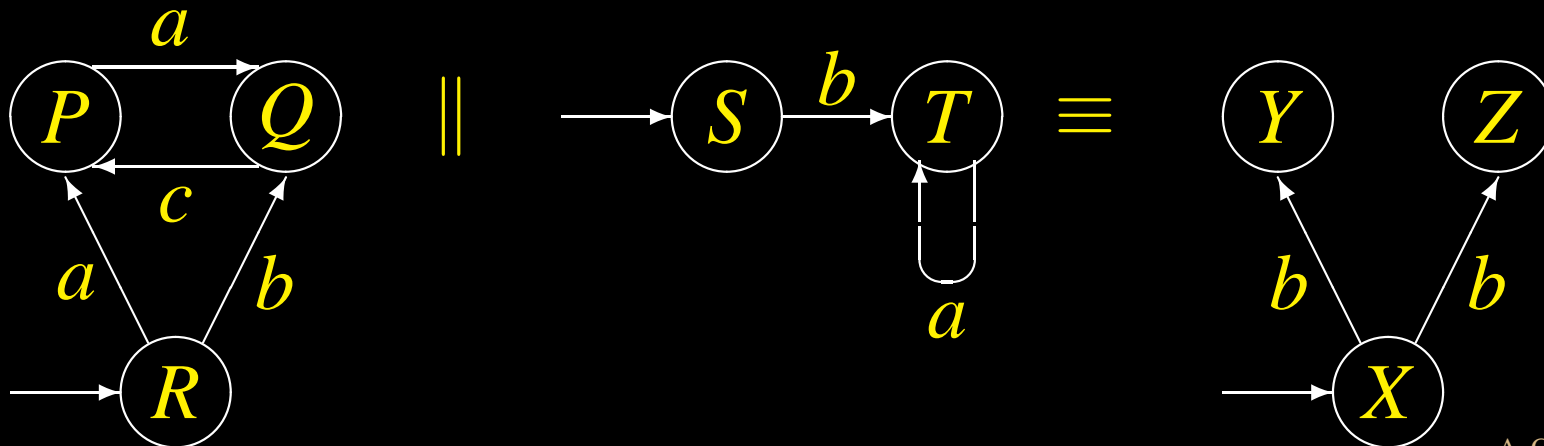
proc X = R [| sync-ab |] S



CSP: Non-Determinism

$$X = R \parallel \{a, c\} \parallel S$$

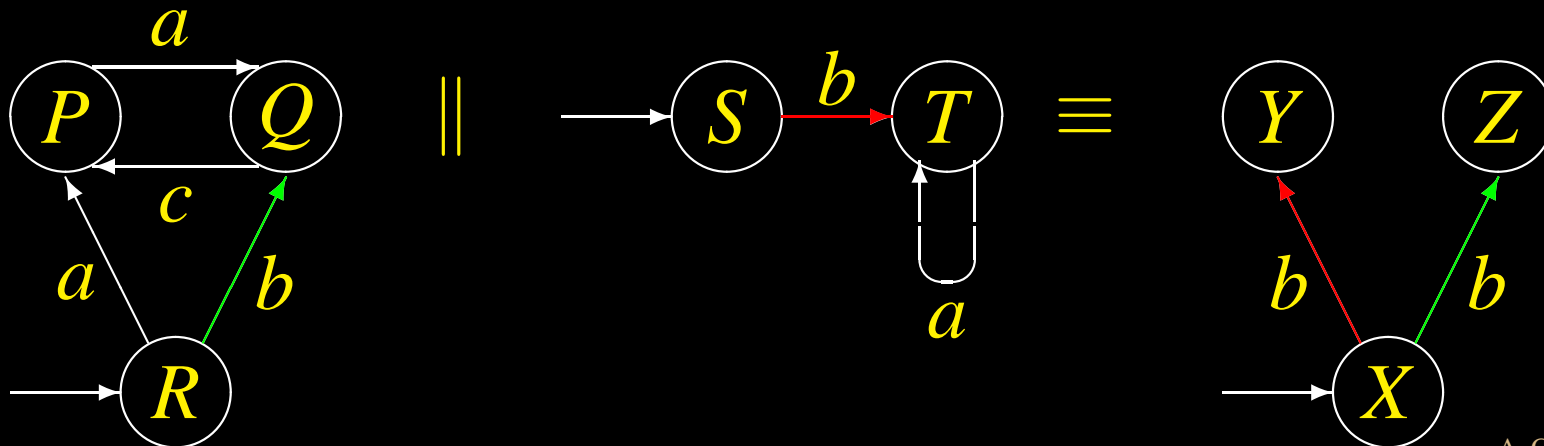
proc Xac = R [| {a, c} |] S



CSP: Non-Determinism

$$X = R \parallel \{a, c\} \parallel S$$

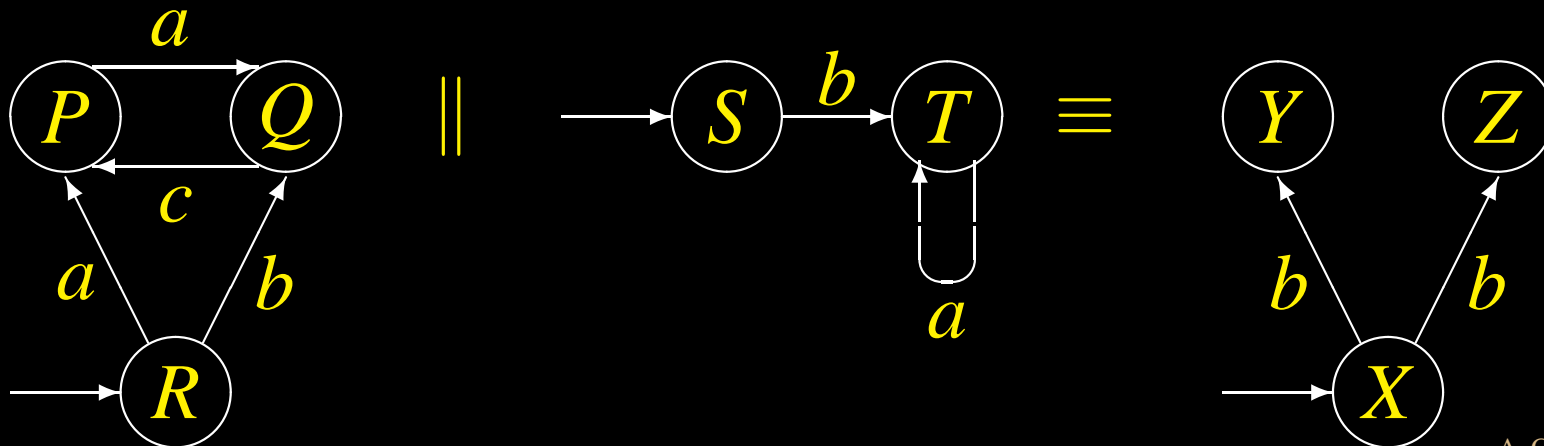
proc Xac = R [| {a, c} |] S



CSP: Deadlock

$$X = R[| \{a, c\} |]S$$

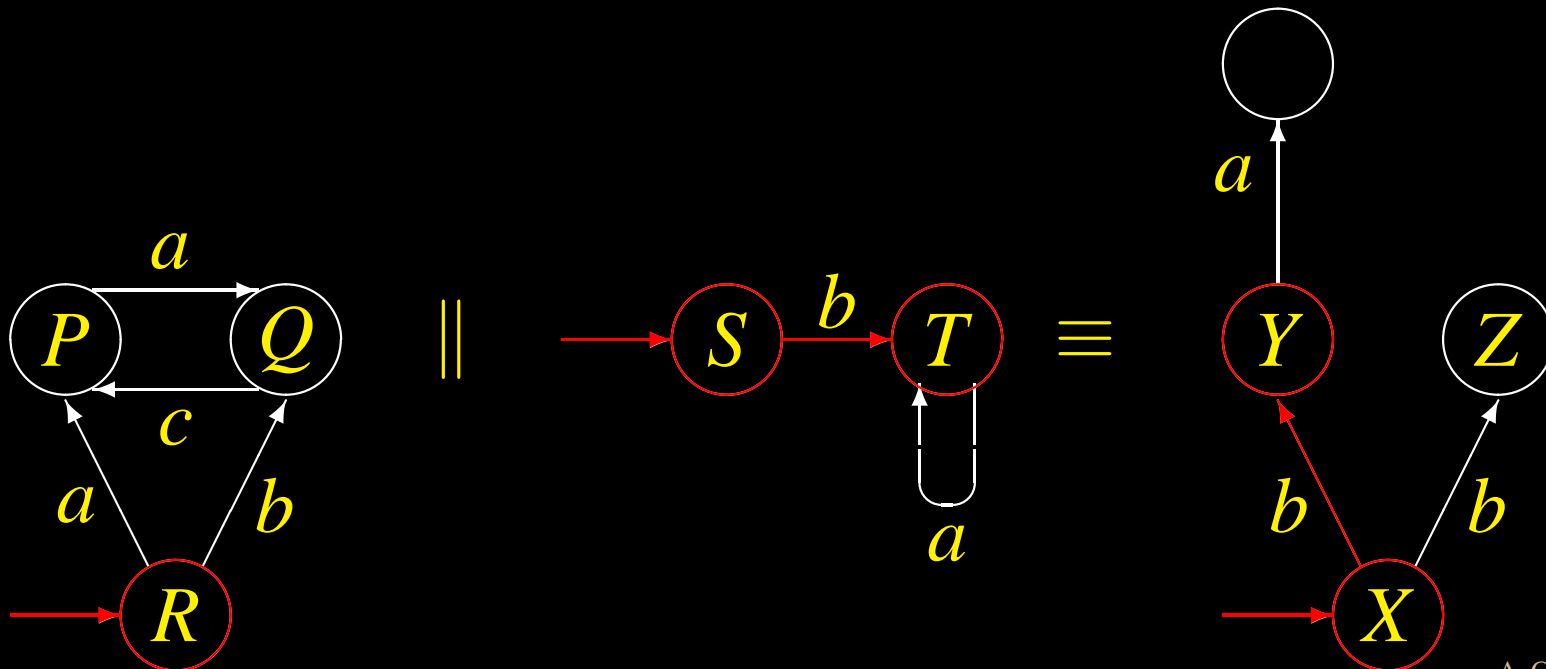
proc Xac = R [| {a, c} |] S



CSP: Deadlock

$$X = R[| \{a, c\} |] S$$

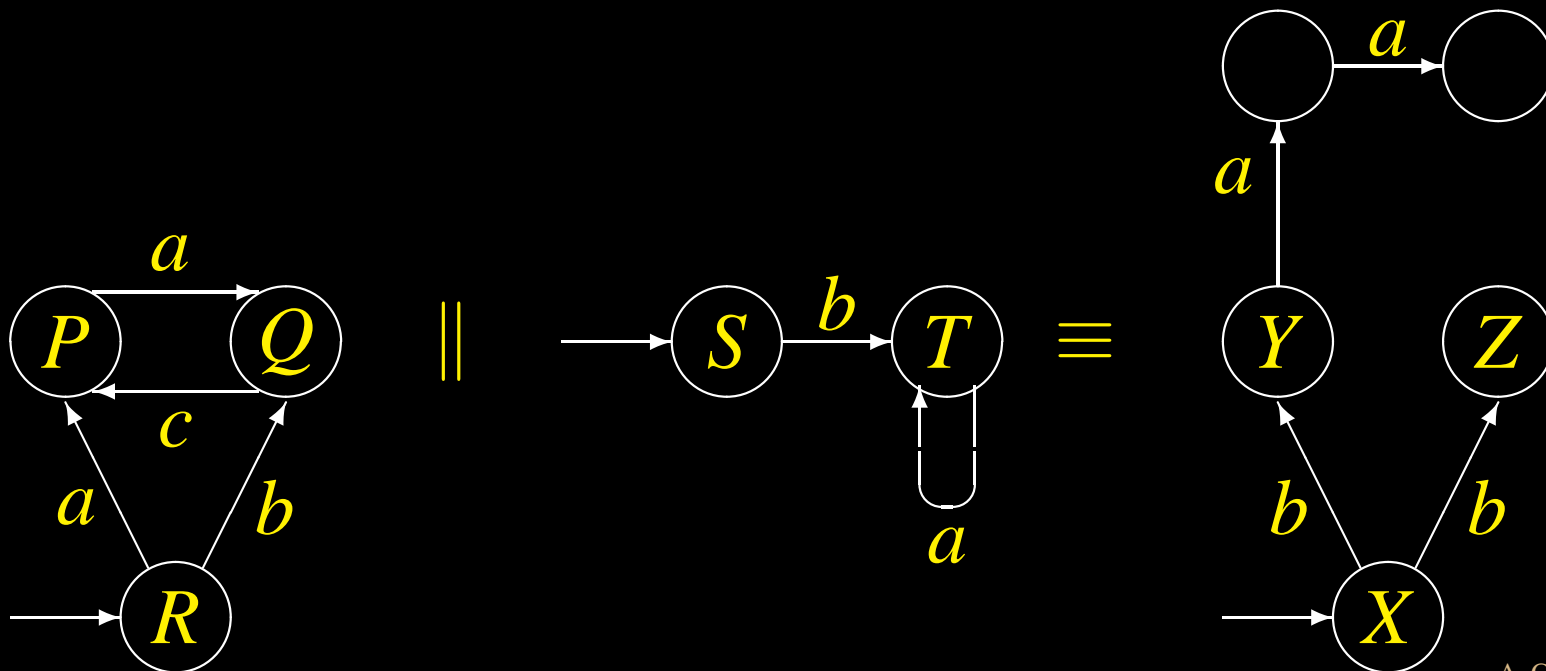
proc Xac = R [| {a, c} |] S



CSP: Deadlock

$$X = R[| \{a, c\} |] S$$

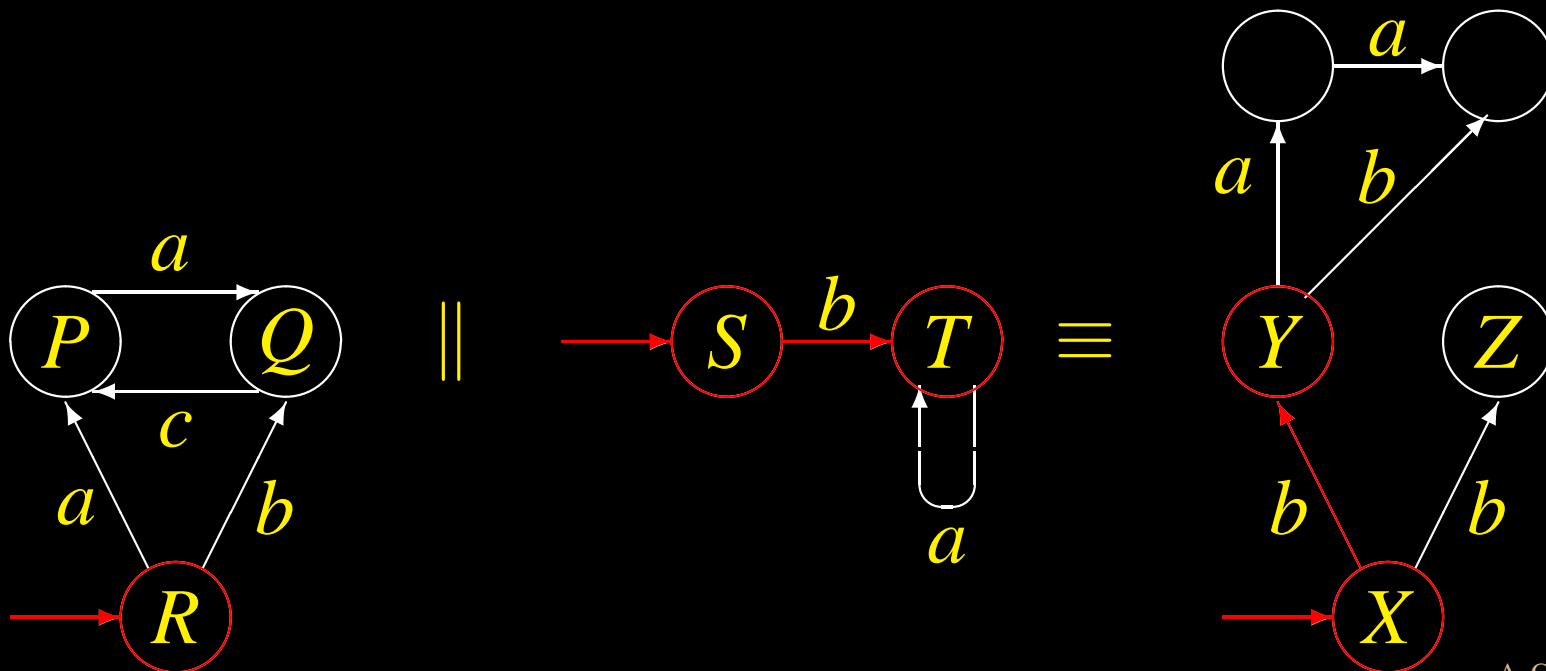
proc Xac = R [| {a, c} |] S



CSP: Deadlock

$$X = R[| \{a, c\} |] S$$

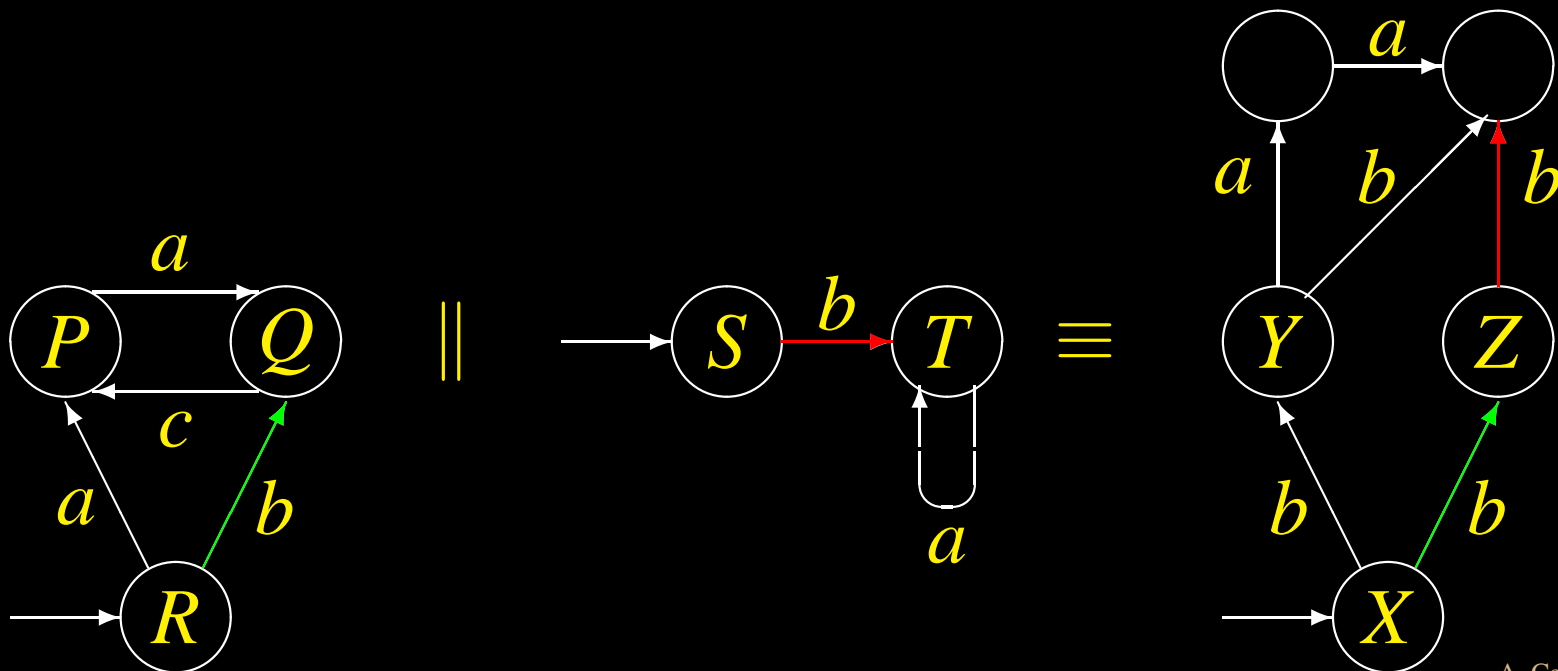
proc Xac = R [| {a, c} |] S



CSP: Deadlock

$$X = R[| \{a, c\} |] S$$

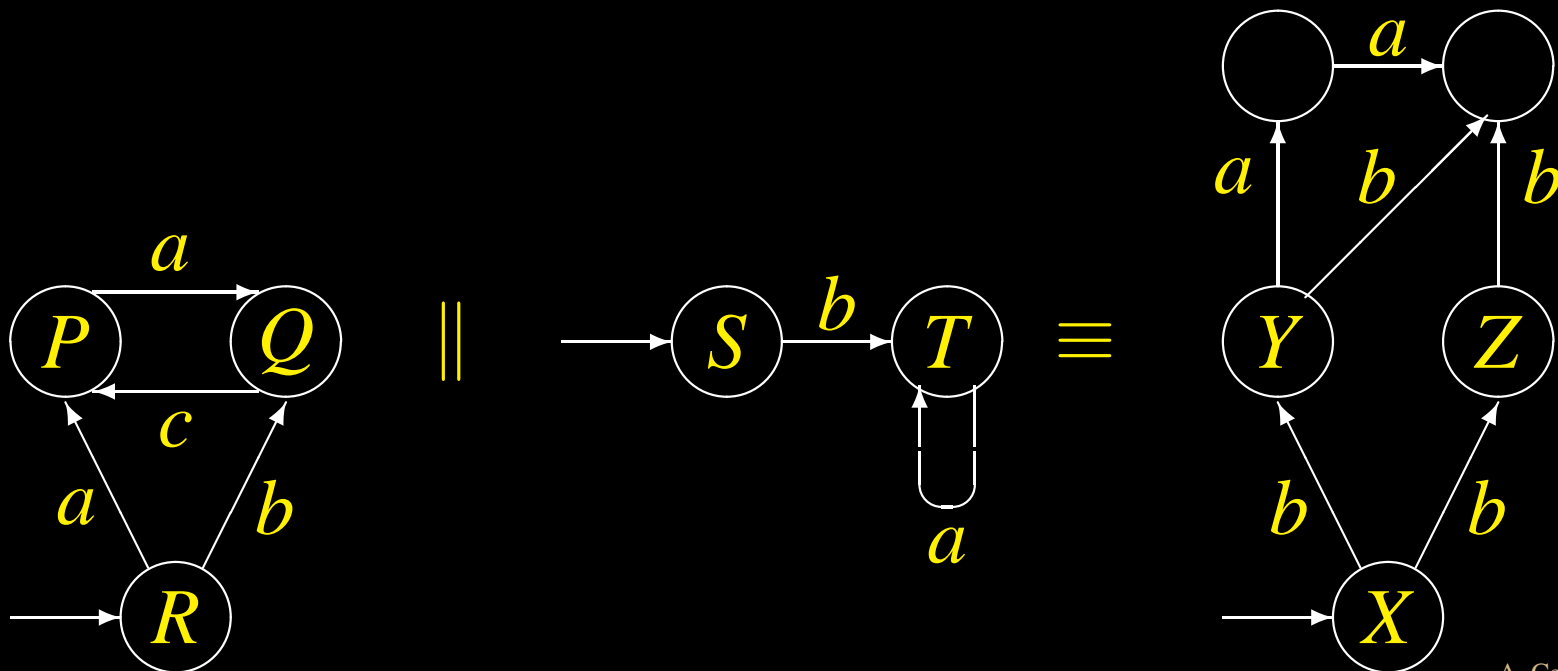
proc Xac = R [| {a, c} |] S



CSP: Deadlock

$$X = R[| \{a, c\} |] S$$

proc Xac = R [| {a, c} |] S



ATM Example

Example: ATM Machine

Informal Specification

An ATM machine requires a user to

- insert a bank card;
- enter the right pin for that card

Example: ATM Machine

Informal Specification

An ATM machine requires a user to

- insert a bank card;
- enter the right pin for that card

Then the machine.

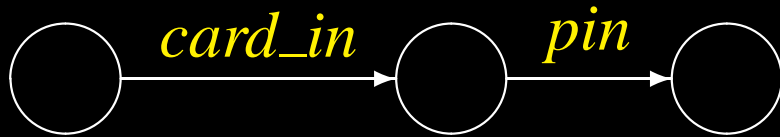
- delivers the cash to the user;
- returns the bank card to the user;
- waits that the user has collected cash and card before being ready for a new transaction.

Example: ATM Machine



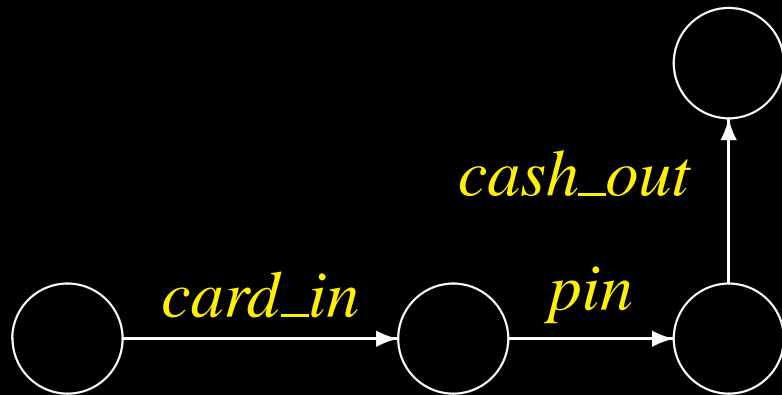
insert a bank card

Example: ATM Machine



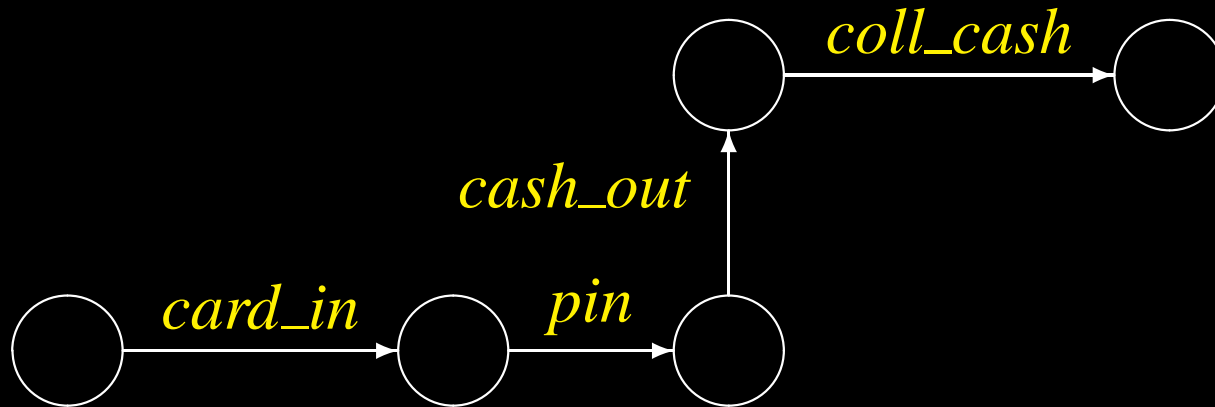
enter the right pin for that card

Example: ATM Machine



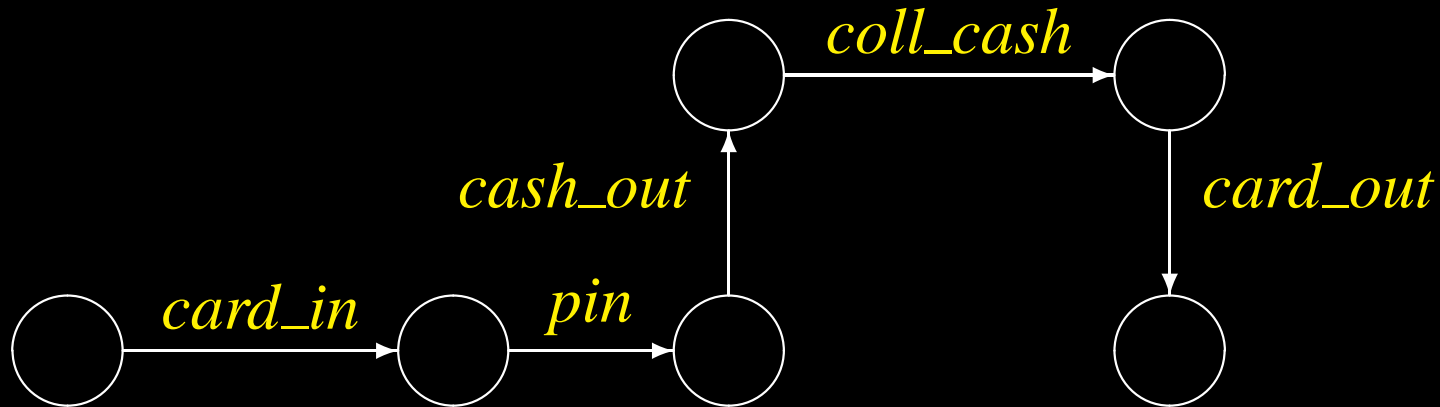
delivers the cash to the user

Example: ATM Machine



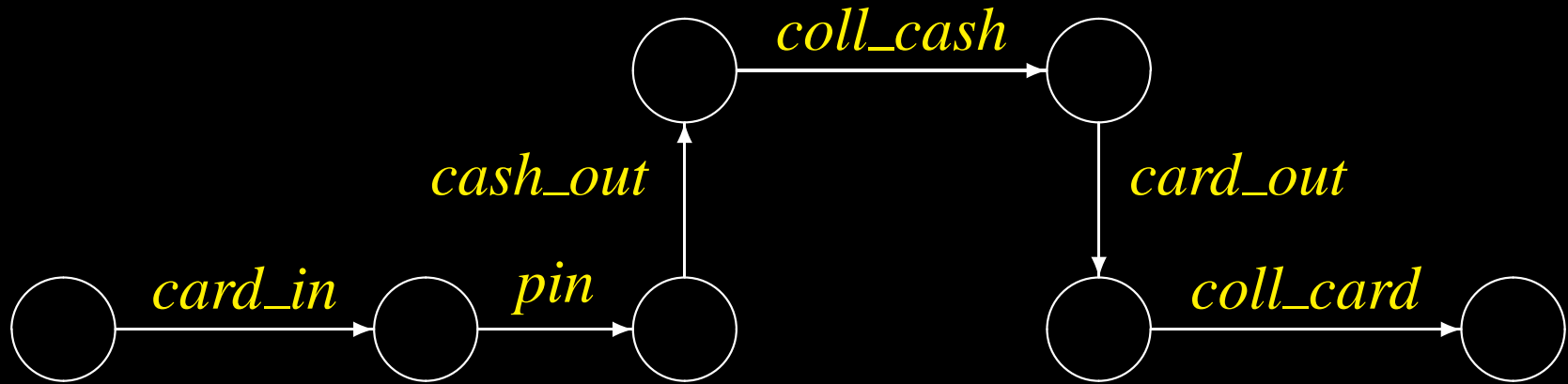
waits that the user has collected cash

Example: ATM Machine



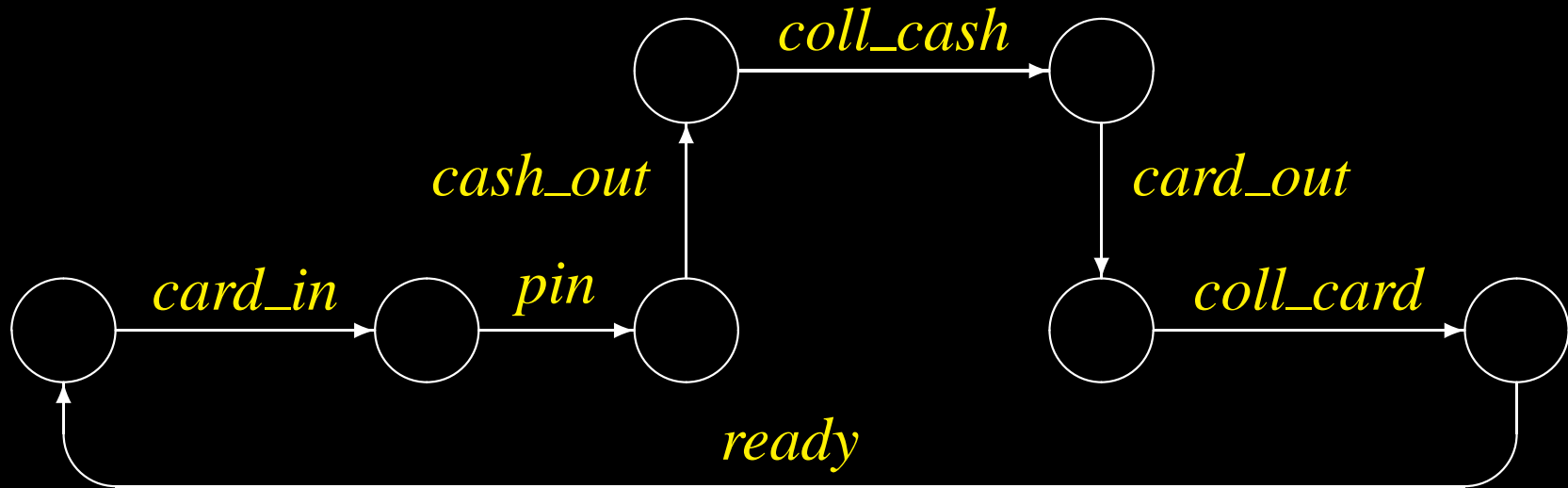
returns the bank card to the user

Example: ATM Machine



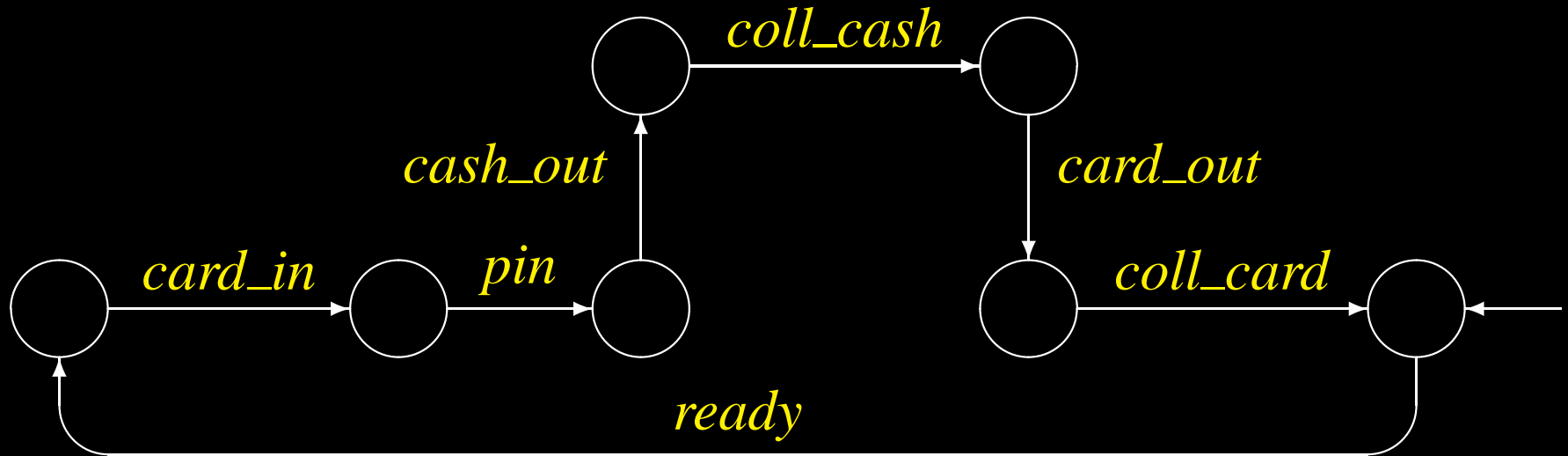
waits that the user has collected card

Example: ATM Machine



be ready for a new transaction

Example: ATM Machine



ATM: CWB-NC Code

CSP Model

```
proc Machine = ready -> card_in  
    -> pin -> cash_out -> CashGiven
```

ATM: CWB-NC Code

CSP Model

```
proc Machine = ready -> card_in  
    -> pin -> cash_out -> CashGiven
```

```
proc CashGiven = coll_cash  
    -> card_out -> CardReturned
```

ATM: CWB-NC Code

CSP Model

```
proc Machine = ready -> card_in  
    -> pin -> cash_out -> CashGiven
```

```
proc CashGiven = coll_cash  
    -> card_out -> CardReturned
```

```
proc CardReturned = coll_card -> Machine
```

Simulation with CWB-NC

...

`cwn-nc>`

Simulation with CWB-NC

...

```
cwn-nc> load atm-machine.csp
```

```
Execution time (user,system,gc,real):(0
```

```
cwn-nc>
```

Simulation with CWB-NC

...

```
cwn-nc> load atm-machine.csp
```

```
Execution time (user,system,gc,real):(0
```

```
cwn-nc> sim Machine
```

```
Machine
```

```
1:  -- ready --> card_in->cash_out->Cash
```

```
cwb-nc-sim>
```

Simulation with CWB-NC

...

```
cwn-nc> load atm-machine.csp
```

```
Execution time (user,system,gc,real):(0
```

```
cwn-nc> sim Machine
```

```
Machine
```

```
1:  -- ready --> card_in->cash_out->Cash
```

```
cwb-nc-sim> 1
```

```
card_in->cash_out->CashGiven
```

```
1:  -- card_in --> cash_out->CashGiven
```

```
cwb-nc-sim>
```

Simulation with CWB-NC

...

```
cwn-nc> load atm-machine.csp
```

```
Execution time (user,system,gc,real):(0
```

```
cwn-nc> sim Machine
```

```
Machine
```

```
1:  -- ready --> card_in->cash_out->Cash
```

```
cwb-nc-sim> 1
```

```
card_in->cash_out->CashGiven
```

```
1:  -- card_in --> cash_out->CashGiven
```

```
cwb-nc-sim> quit
```

```
Execution time (user,system,gc,r
```

```
cwb-nc>
```


Simulation with CWB-NC

...

```
cwn-nc> load atm-machine.csp
```

```
Execution time (user,system,gc,real):(0
```

```
cwn-nc> sim Machine
```

```
Machine
```

```
1:  -- ready --> card_in->cash_out->Cash
```

```
cwb-nc-sim> 1
```

```
card_in->cash_out->CashGiven
```

```
1:  -- card_in --> cash_out->CashGiven
```

```
cwb-nc-sim> quit
```

```
Execution time (user,system,gc,r
```

```
cwb-nc> quit
```

```
shell>
```

ATM Machine: Exercises

Modify the ATM

1. to allow a customer to choose between
 - cash withdrawal, and
 - statements printing

ATM Machine: Exercises

Modify the ATM

1. to allow a customer to choose between
 - cash withdrawal, and
 - statements printing
2. to take back card and/or cash if they are not collected by the customer within a given time

ATM Machine: Exercises

Modify the ATM

1. to allow a customer to choose between
 - cash withdrawal, and
 - statements printing
2. to take back card and/or cash if they are not collected by the customer within a given time
3. $1 + 2$

ATM Specification

Informal Specification

An ATM machine **requires** a user to

- insert a bank card
- enter the right pin for that card

Then the machine

- delivers the cash to the user
- returns the bank card to the user
- waits that the user has collected cash and card before being ready for a new transaction.

ATM Spec: “requires” part

cash delivered to the user

requires

bank card inserted

and

right pin for that card entered

ATM Spec: “requires” part

cash delivered to the user

requires

bank card inserted

and

right pin for that card entered

cash_out requires (*card_in* and *pin*)

ATM Spec: “requires” part

cash delivered to the user

requires

bank card inserted

and

right pin for that card entered

cash_out requires *card_in*

ATM Spec: “requires” part

cash delivered to the user

requires

bank card inserted

and

right pin for that card entered

cash_out requires *card_in*

((not *cash_out*) until *card_in*)

ATM Spec: “requires” part

cash delivered to the user

requires

bank card inserted

and

right pin for that card entered

cash_out requires *card_in*

((not *cash_out*) until *card_in*)
after the machine is *ready*

ATM Spec: “requires” part

cash delivered to the user

requires

bank card inserted

and

right pin for that card entered

cash_out requires *card_in*

((not *cash_out*) until *card_in*)
after the machine is *ready*

$\forall \square (ready \rightarrow ((\neg cash_out) \mathcal{U} card_in))$

Missing Part

Informal Specification

An ATM machine requires a user to

- insert a bank card
- enter the right pin for that card

Then the machine

- delivers the cash to the user
- returns the bank card to the user
- waits that the user has collected cash and card before being ready for a new transaction.

ATM Spec: “allows” part

if

- bank card inserted
- right pin for that card entered

then

- cash delivered to the user
before the machine is ready again
- card returned to the user
before the machine is ready again
- the user has to collect the cash
before the machine is ready again
- the user has to collect the card
before the machine is ready again

ATM Spec: “allows” part

- if
 bank card inserted and
 right pin for that card entered
then
 cash delivered to the user
 before the machine is ready again
- card returned to the user
 before the machine is ready again
- the user has to collect the cash
 before the machine is ready again
- the user has to collect the card
 before the machine is ready again

ATM Spec: “allows” part

- if
 bank card inserted and later
 right pin for that card entered
or
 right pin for that card entered and later
 bank card inserted
then
 cash delivered to the user
 before the machine is ready again

ATM Spec: “allows” part

- if
 - bank card inserted and later
 - right pin for that card entered
 or
 - right pin for that card entered and later
 - bank card inserted
 then
 - cash delivered to the user
 - before the machine is ready again

$$\begin{aligned} &\forall \square ((card_in \wedge ((\neg ready) \mathcal{U} pin) \\ &\quad \rightarrow ((\neg ready) \mathcal{U} cash_out))) \vee \\ &\quad (pin \wedge ((\neg cash_out) \mathcal{U} card_in) \\ &\quad \rightarrow (\neg ready) \mathcal{U} cash_out)) \end{aligned}$$

ATM Spec: “allows” part

- card returned to the user
before the machine is ready again

ATM Spec: “allows” part

- card returned to the user
before the machine is ready again

$$\forall \square (card_in \rightarrow ((\neg ready) \mathcal{U} card_out))$$

ATM Spec: “allows” part

- card returned to the user
before the machine is ready again
$$\forall \square (card_in \rightarrow ((\neg ready) \mathcal{U} card_out))$$
- the user has to collect the cash
before the machine is ready again

ATM Spec: “allows” part

- card returned to the user
before the machine is ready again

$$\forall \square (card_in \rightarrow ((\neg ready) \mathcal{U} card_out))$$

- the user has to collect the cash
before the machine is ready again

$$\forall \square (cash_out \rightarrow ((\neg ready) \mathcal{U} coll_cash))$$

ATM Spec: “allows” part

- card returned to the user
before the machine is ready again
$$\forall \square (card_in \rightarrow ((\neg ready) \mathcal{U} card_out))$$
- the user has to collect the cash
before the machine is ready again
$$\forall \square (cash_out \rightarrow ((\neg ready) \mathcal{U} coll_cash))$$
- the user has to collect the card
before the machine is ready again

ATM Spec: “allows” part

- card returned to the user
before the machine is ready again
$$\forall \square (card_in \rightarrow ((\neg ready) \mathcal{U} card_out))$$
- the user has to collect the cash
before the machine is ready again
$$\forall \square (cash_out \rightarrow ((\neg ready) \mathcal{U} coll_cash))$$
- the user has to collect the card
before the machine is ready again
$$\forall \square (card_out \rightarrow ((\neg ready) \mathcal{U} coll_card))$$

Model-checking



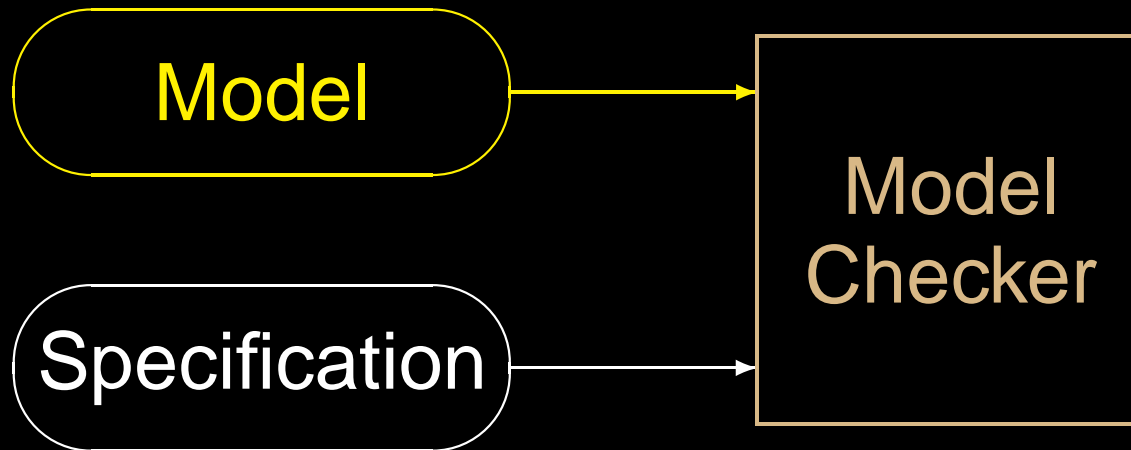
Model

Model-checking

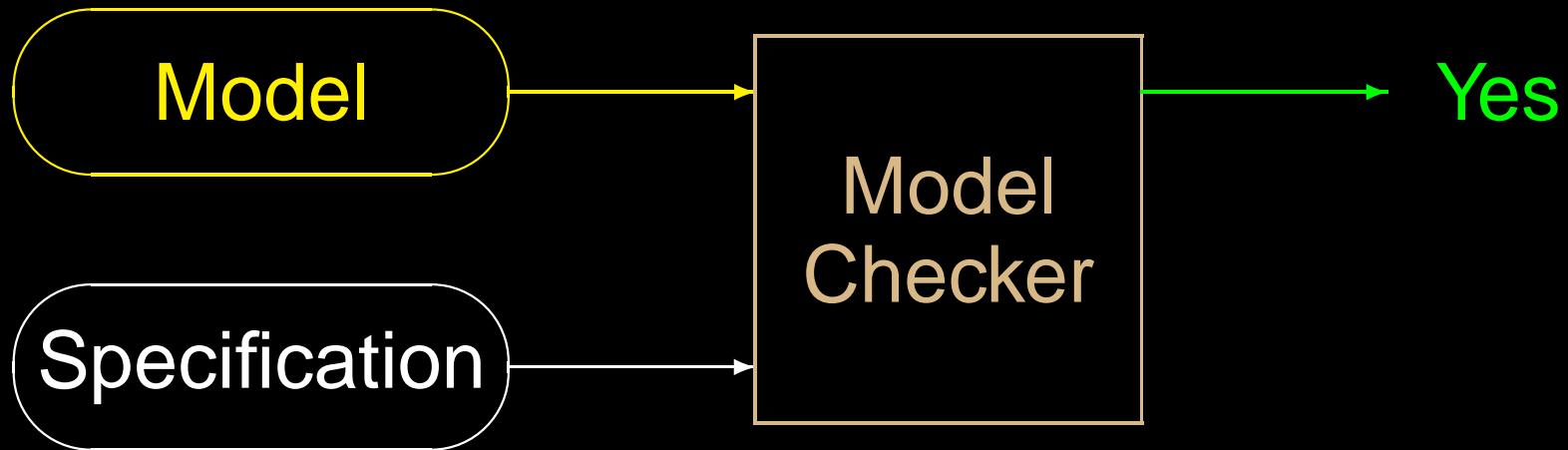
Model

Specification

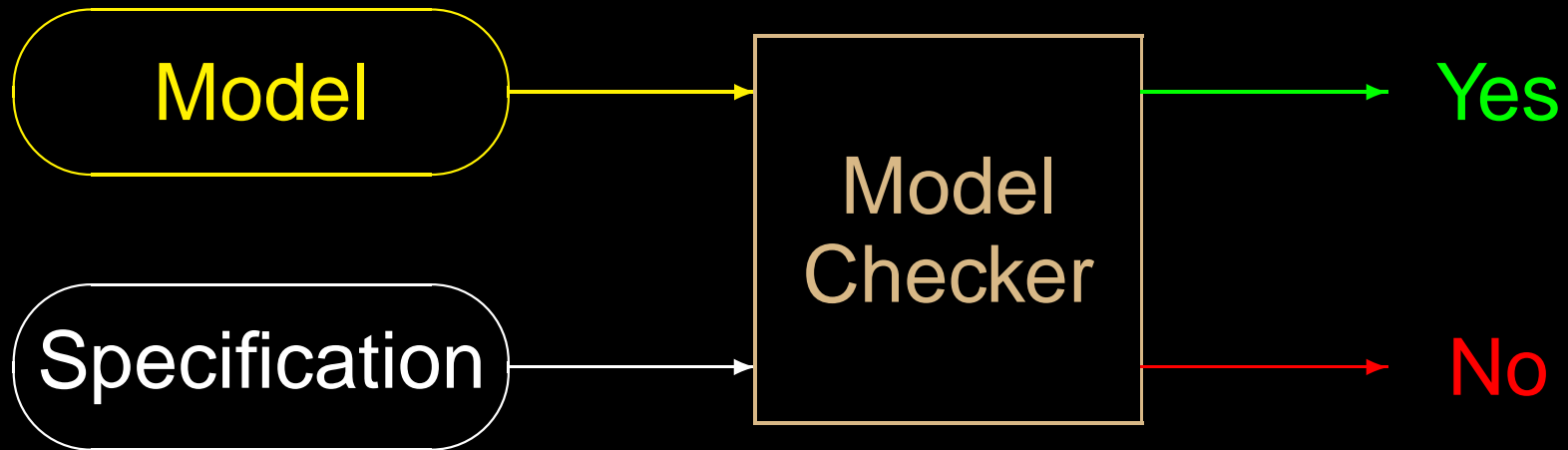
Model-checking



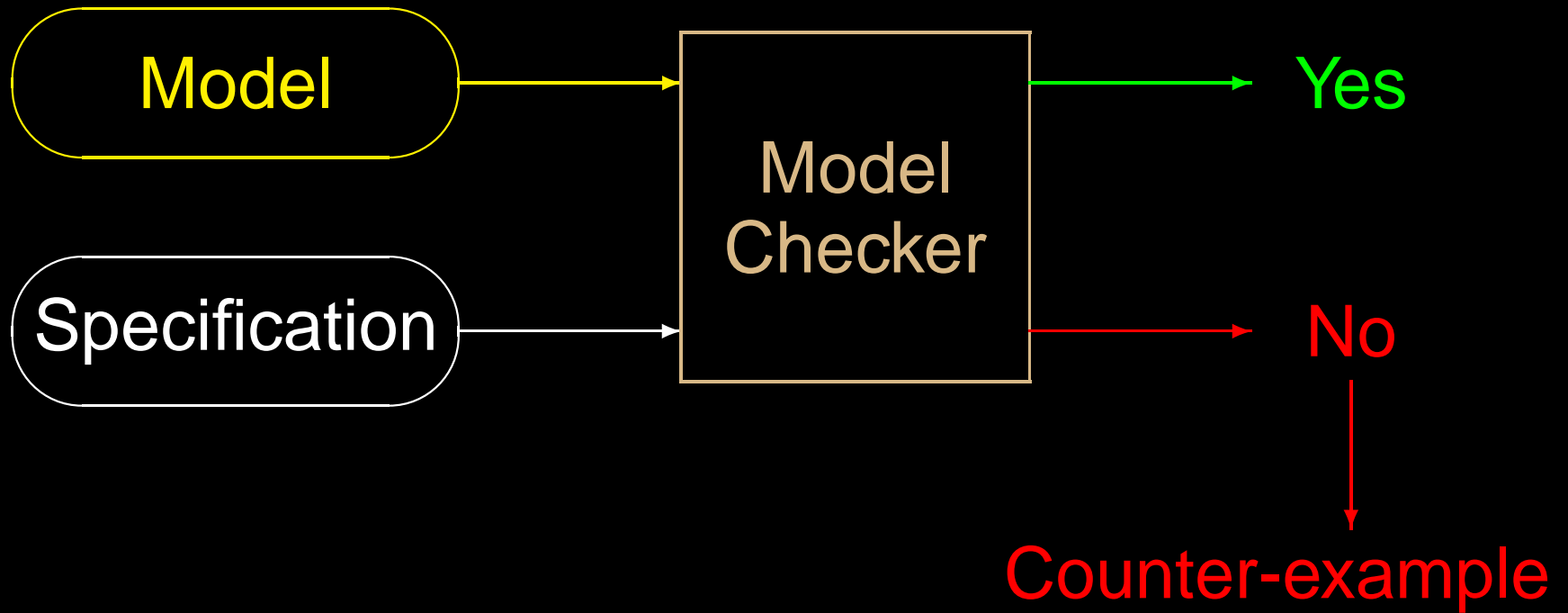
Model-checking



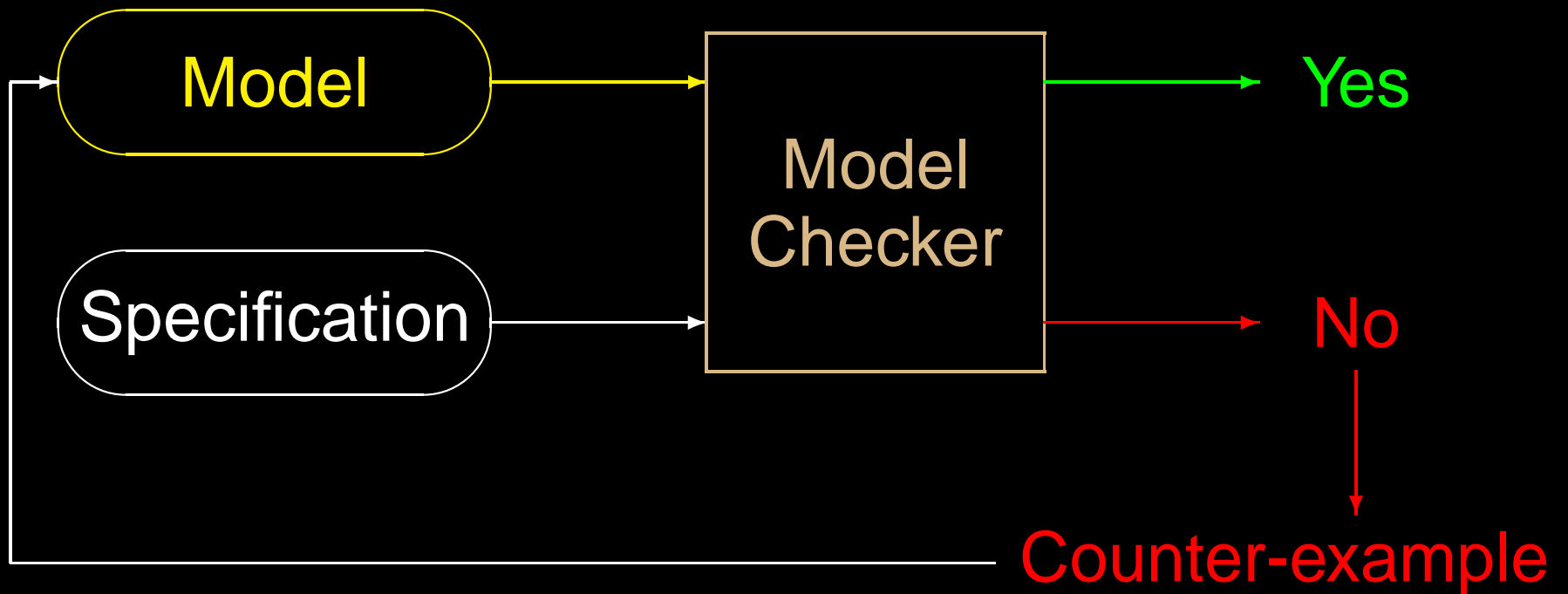
Model-checking



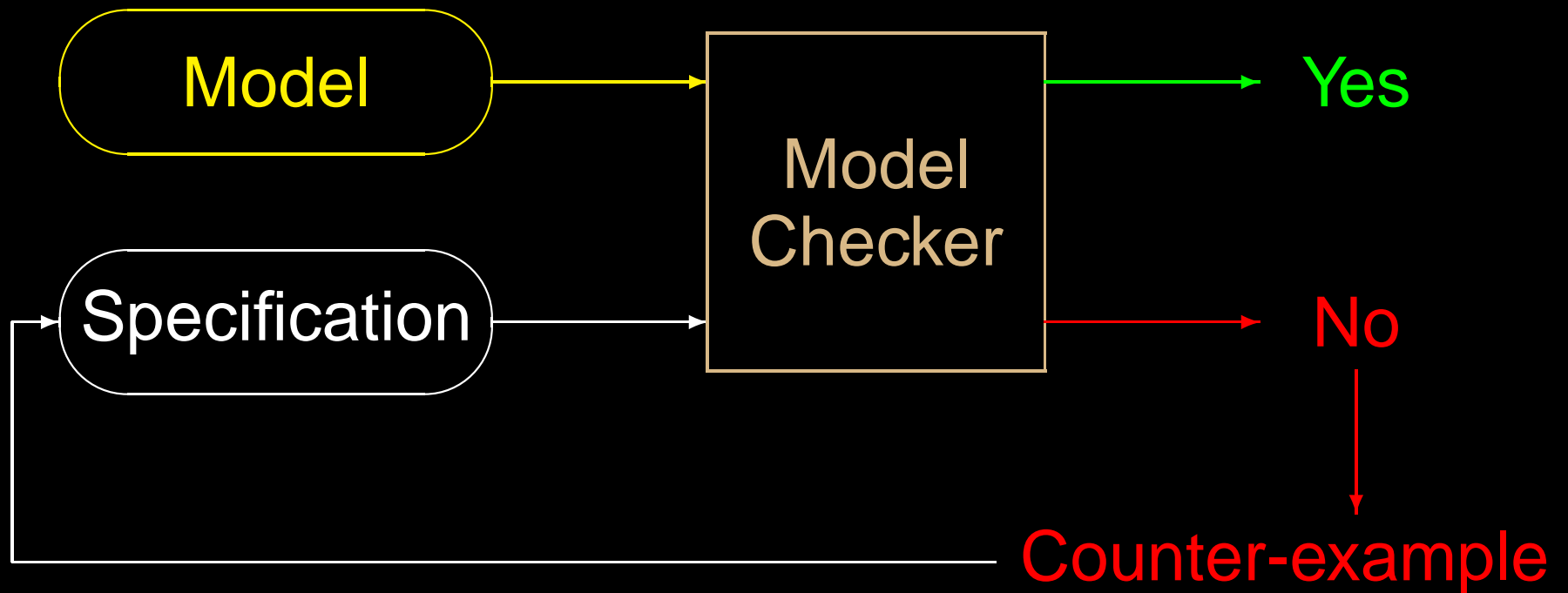
Model-checking



Model-checking



Model-checking



HCI Concepts

HCI and Interactive Systems

Humans (Users) interact with Computers

- to achieve goals
- by performing tasks

HCI and Interactive Systems

Humans (Users) interact with Computers

- to achieve goals
- by performing tasks

Interactive Systems are designed to assist user

User: first priority in the requirements

HCI and Interactive Systems

Humans (Users) interact with Computers

- to achieve goals
- by performing tasks

Interactive Systems are designed to assist user

User: first priority in the requirements

Need to understand

- capabilities
- limitations

of the user

Relevant Human Aspects

(which have a bearing with Computer Systems)

- how humans **perceive** the world around them
- how they **store information** and **solve problems**
- how they **physically manipulate objects**

Relevant Human Aspects

(which have a bearing with Computer Systems)

- how humans **perceive** the world around them
- how they **store information** and **solve problems**
- how they **physically manipulate objects**

⇒ (simplified) model of human processing

Relevant Human Aspects

(which have a bearing with Computer Systems)

- how humans **perceive** the world around them
- how they **store information** and **solve problems**
- how they **physically manipulate objects**

⇒ (simplified) model of human processing
based on

- **Computer Analogy**
- **Information Processing Theory**

Computer Analogy

Computers take a symbolic input, recode it, make decisions about the recoded input, make new expressions from it, store some or all of the input, and give back a symbolic input.

By analogy that is what most cognitive psychology is about.

It is about how most people take in information, how they recode and remember it, how they make decisions, how they transform their internal knowledge states, and how they translate these states into behavioural outputs.

[Lachman et al. 79]

R. Lachman, J. L. Lachman, E. C. Butterfield.

Cognitive Psychology and Information Processing.

Lawrence Erlbaum, 1979.

Organisational Level Analogy

- Central Processing Unit analogous to the mechanism responsible for **mental operations to manipulate information**
- Information Store analogous to **long-term memory**
- Information Buffer analogous to **short-term memory**

Organisational Level Analogy

- Central Processing Unit analogous to the mechanism responsible for **mental operations to manipulate information**
- Information Store analogous to **long-term memory**
- Information Buffer analogous to **short-term memory**

Unlikely computers **humans** are also **influenced by external factors**, such as social and organisational environment.

Information Processing Theory

- **Behaviourism:** Psychology should be based solely on observable events, with no mentalistic concepts

Information Processing Theory

- **Behaviourism:** Psychology should be based solely on observable events, with no mentalistic concepts
- **Information Processing** defines models to characterise the nature of mental processes

Information Processing Theory

- **Behaviourism:** Psychology should be based solely on observable events, with no mentalistic concepts
- **Information Processing** defines models to characterise the nature of mental processes
 - **based on computer analogy**

Information Processing Theory

- **Behaviourism:** Psychology should be based solely on observable events, with no mentalistic concepts
 - **Information Processing** defines models to characterise the nature of mental processes
 - based on computer analogy
 - use experiments based on
 - analysis of response
 - subjective analysis
- to confirm and extend the theory

Model Human Processor

developed by Card, Moran and Newell in 1983
[Card et al. 83], consists of:

- perceptual system handling sensory stimulus from the outside world
- motor system which control actions
- cognitive system which connects the other two subsystems

Model Human Processor

developed by Card, Moran and Newell in 1983
[Card et al. 83], consists of:

- perceptual system handling sensory stimulus from the outside world
- motor system which control actions
- cognitive system which connects the other two subsystems

each equipped with its own processor and memory (short-term and long-term).

Model Human Processor

developed by Card, Moran and Newell in 1983
[Card et al. 83], consists of:

- perceptual system handling sensory stimulus from the outside world
- motor system which control actions
- cognitive system which connects the other two subsystems

each equipped with its own processor and memory (short-term and long-term). In addition

- principles of operation dictates the behaviour of the system under certain conditions

Simplified Generic Model

A **human system** is an intelligent information processing system

Simplified Generic Model

A **human system** is an intelligent information processing system consisting of:

- **Input-Output:** senses and responders (or effectors)
involves some low-level processing

Simplified Generic Model

A **human system** is an intelligent information processing system consisting of:

- **Input-Output:** senses and responders (or effectors)
involves some low-level processing
- **Memory** (short-term and long-term)

Simplified Generic Model

A **human system** is an intelligent information processing system consisting of:

- **Input-Output:** senses and responders (or effectors)
involves some low-level processing
- **Memory** (short-term and long-term)
- **Processing**
 - problem solving
 - learning

Simplified Generic Model

A **human system** is an intelligent information processing system consisting of:

- **Input-Output:** senses and responders (or effectors)
involves some low-level processing
- **Memory** (short-term and long-term)
- **Processing**
 - problem solving
 - learning and consequently
 - making mistakes

User Knowledge

- goal

User Knowledge

- goal
- about task
 - actions to perform it

User Knowledge

- goal
- about task
 - actions to perform it
 - (possibly) structure of the set of actions

User Knowledge

- goal
 - about task
 - actions to perform it
 - (possibly) structure of the set of actions
- independently of a specific
computer/machine/interface

User Knowledge

- goal
- about task
 - actions to perform it
 - (possibly) structure of the set of actionsindependently of a specific computer/machine/interface
- about machine
 - expertise acquired through use/training

User Knowledge

- goal
- about task
 - actions to perform it
 - (possibly) structure of the set of actionsindependently of a specific computer/machine/interface
- about machine
 - expertise acquired through use/training
 - mental model

User Actions

- **mental processing** structured set of actions

User Actions

- mental processing structured set of actions
- interaction
 - driven by the mental model
 - triggered by the machine

User Actions

- mental processing structured set of actions
- interaction
 - driven by the mental model
 - triggered by the machine

involve the use of human memory

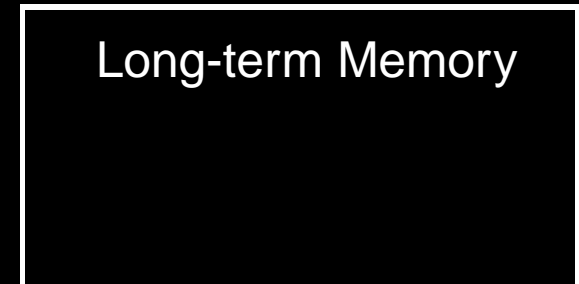
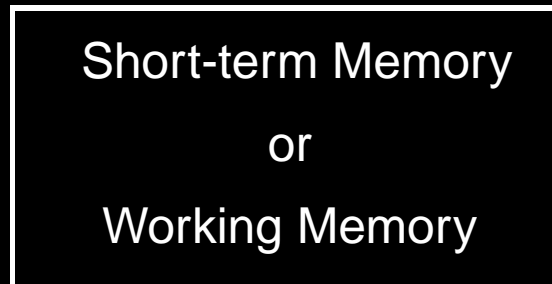
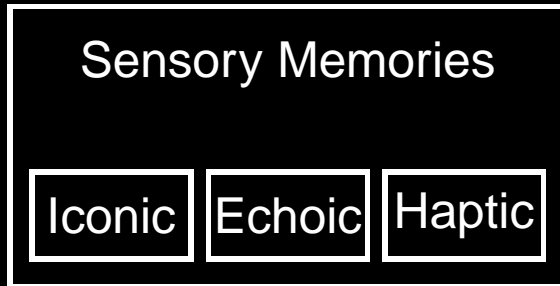
Human Memory

Sensory Memories

Short-term Memory
or
Working Memory

Long-term Memory

Human Memory



Human Memory

Sensory Memories

Iconic

Echoic

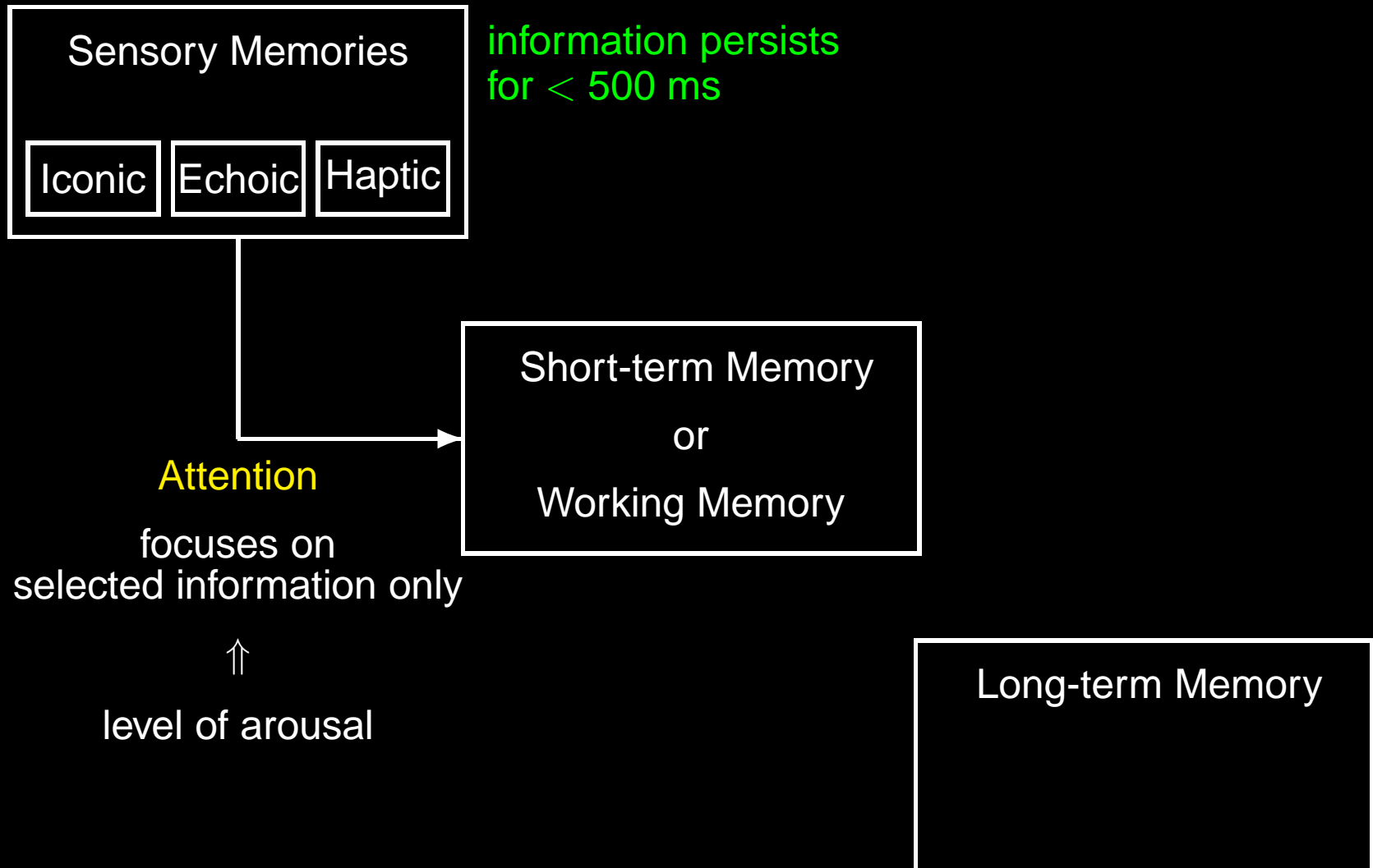
Haptic

information persists
for < 500 ms

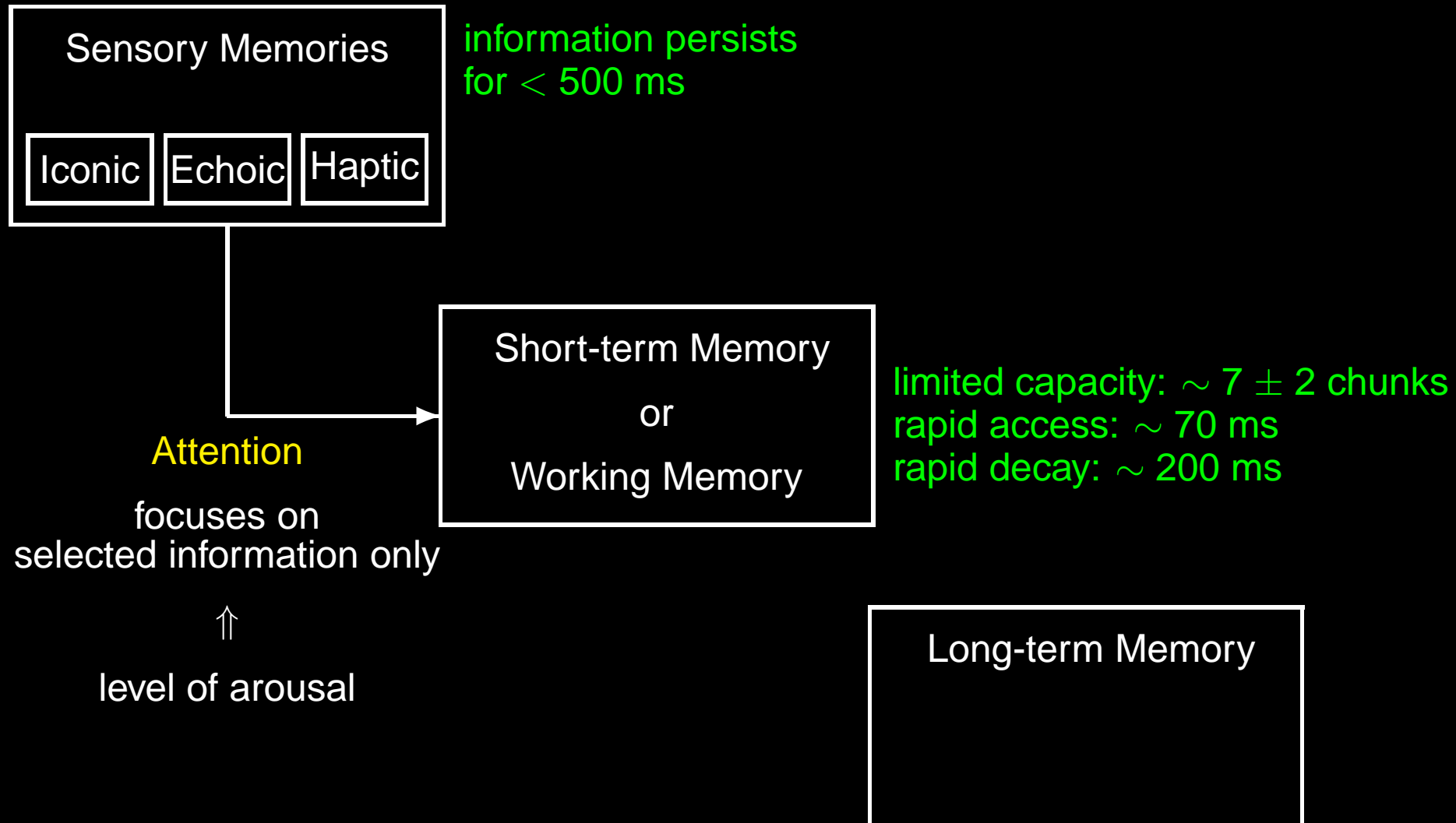
Short-term Memory
or
Working Memory

Long-term Memory

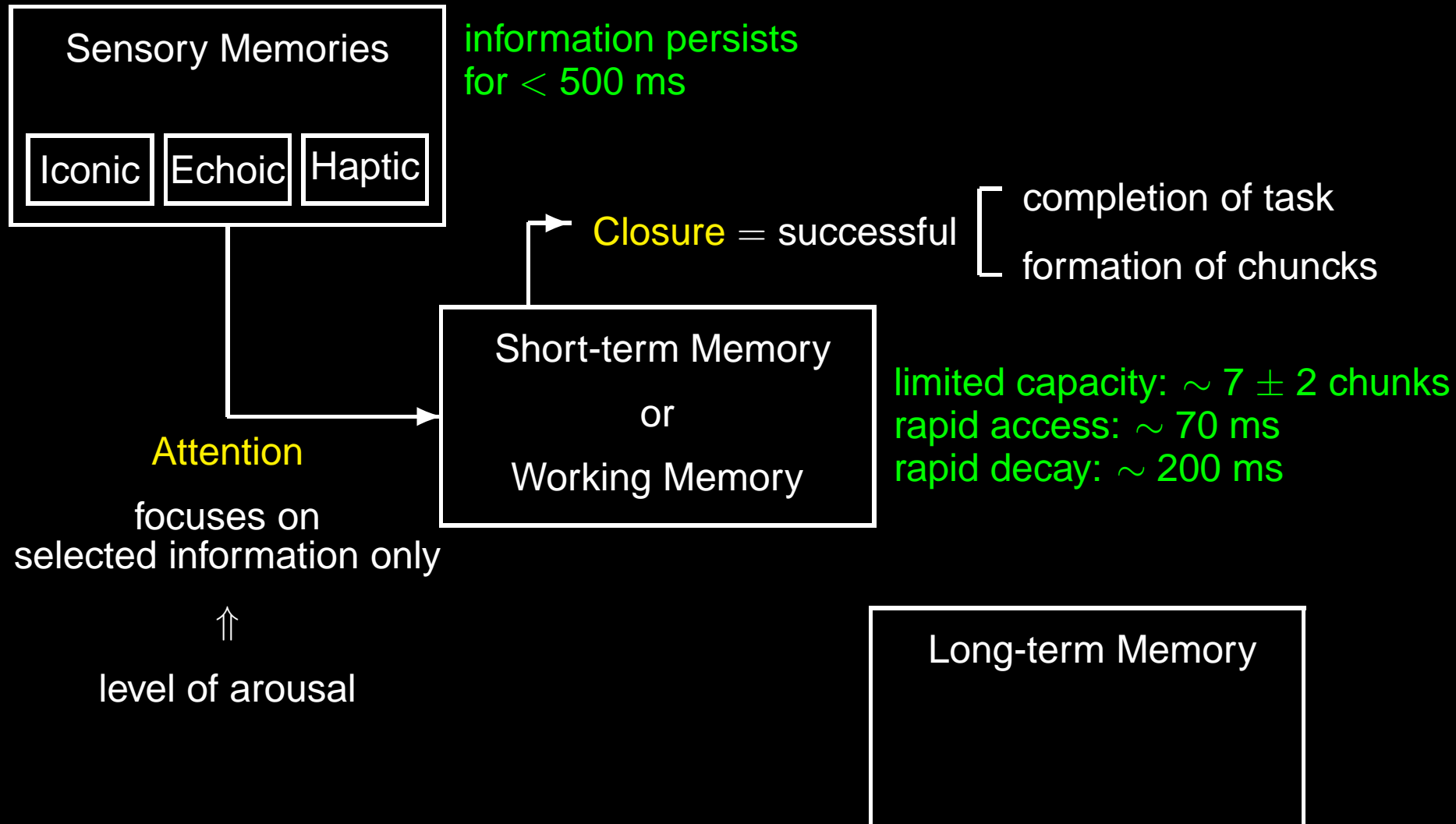
Human Memory



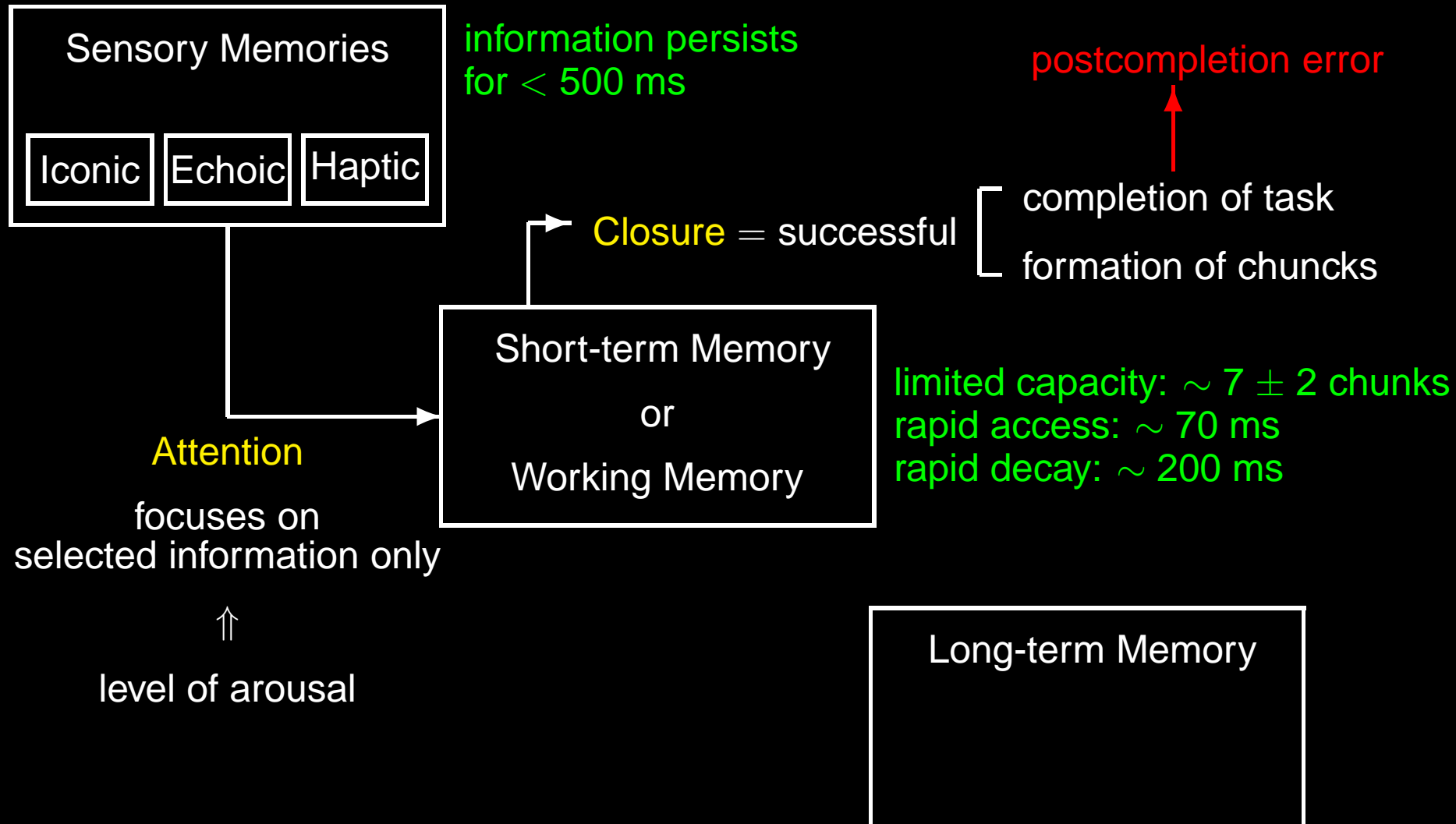
Human Memory



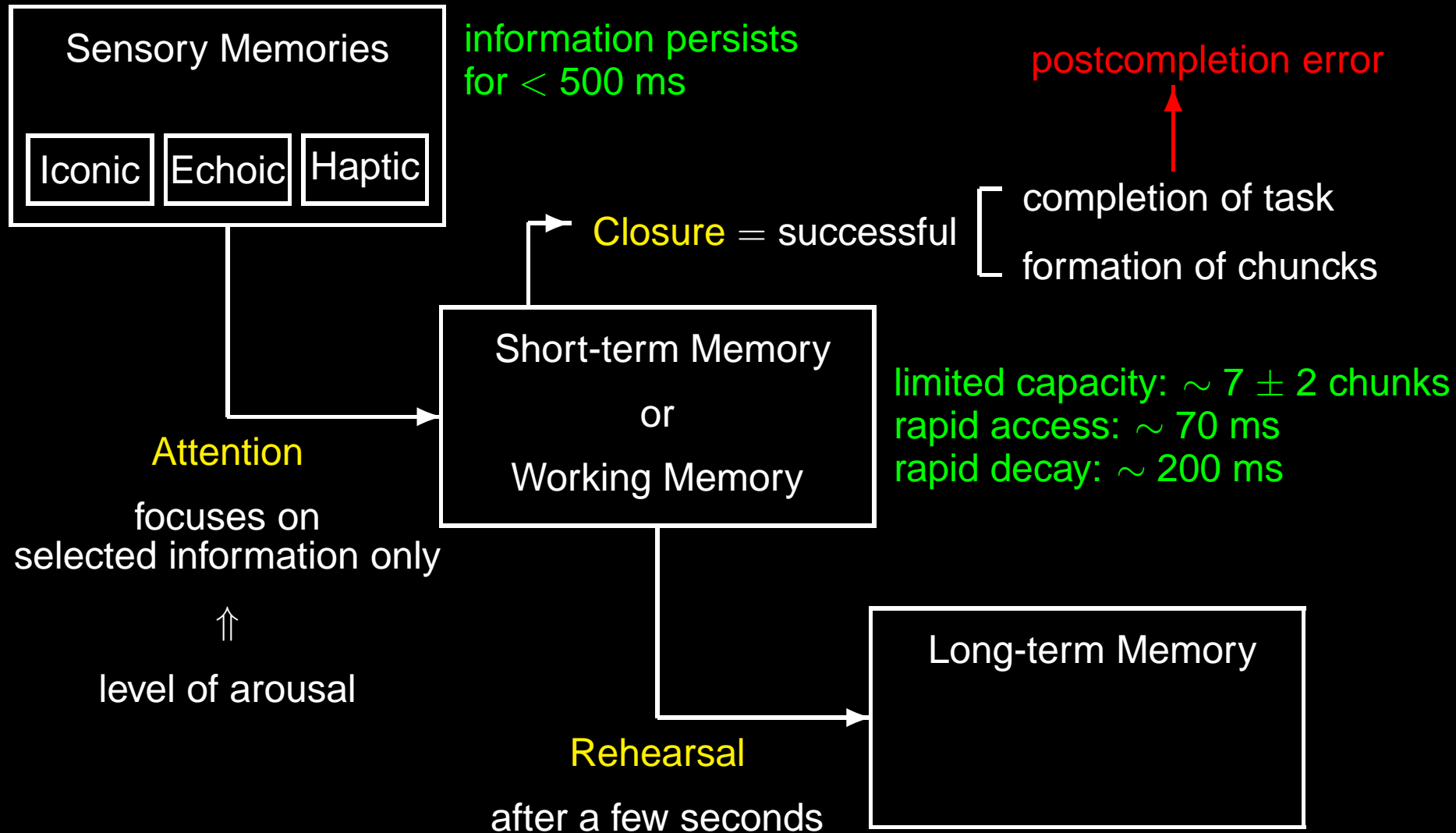
Human Memory



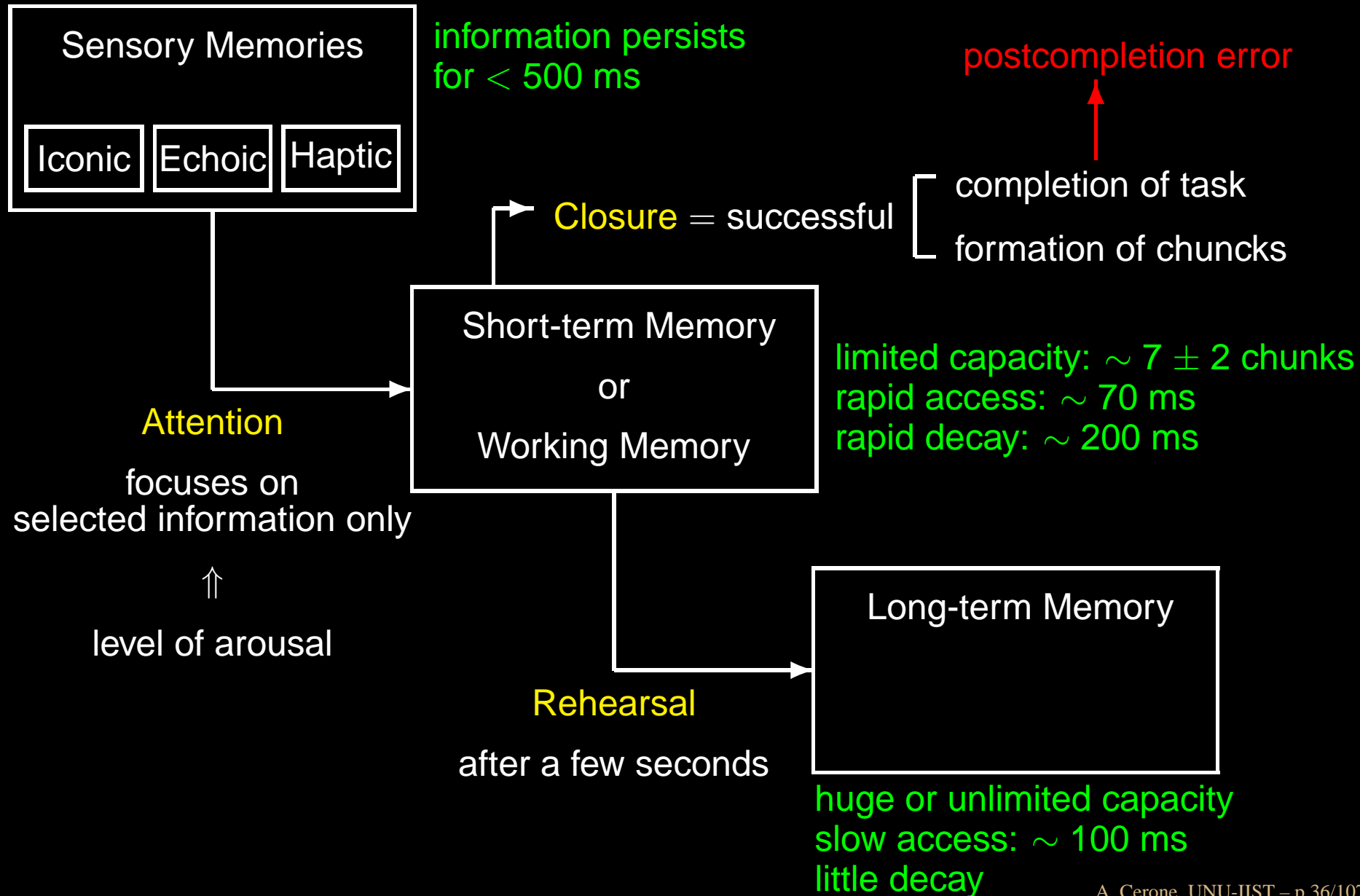
Human Memory



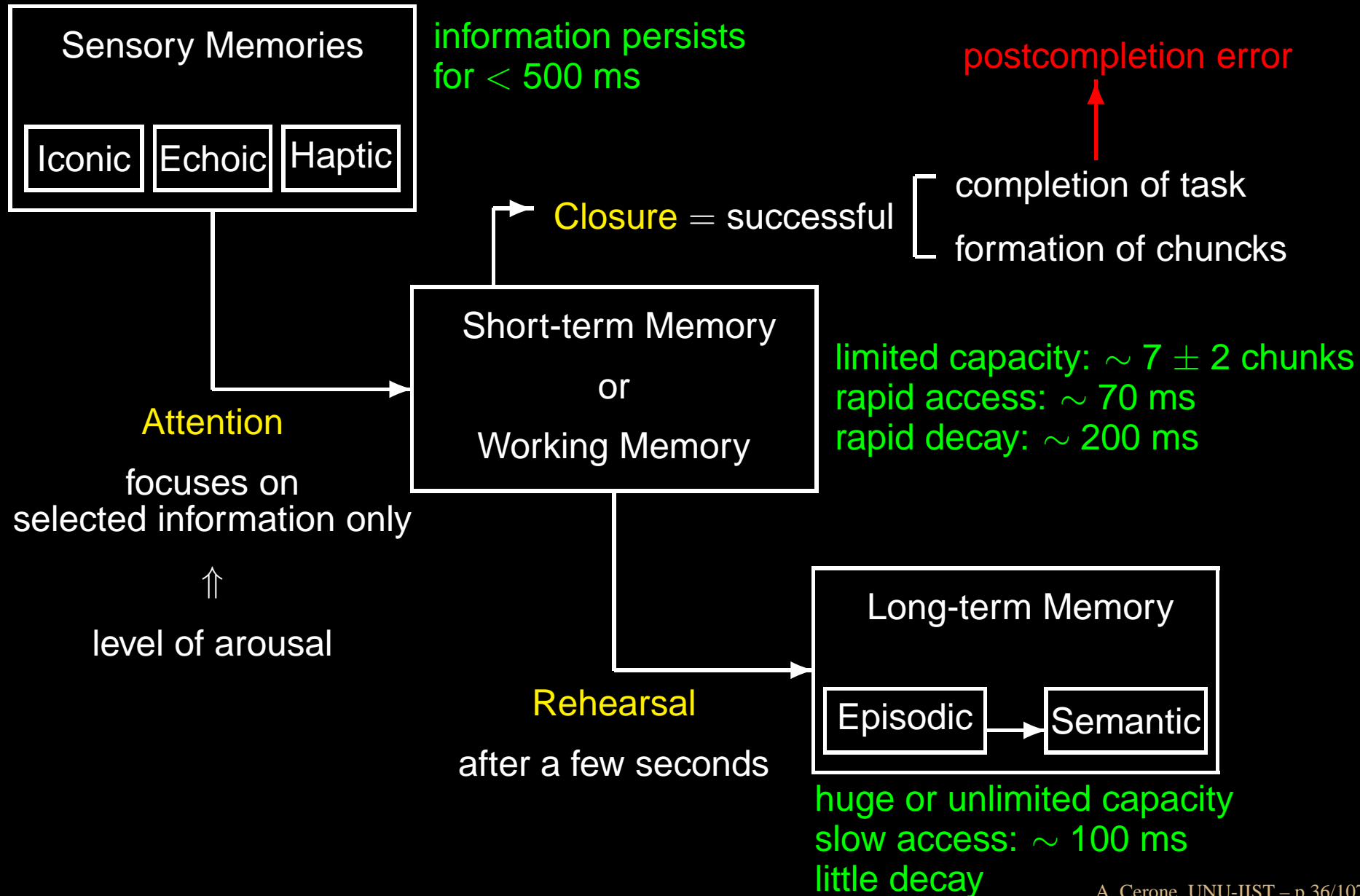
Human Memory



Human Memory



Human Memory



Modelling Human Behaviour

ATM Example Revisited

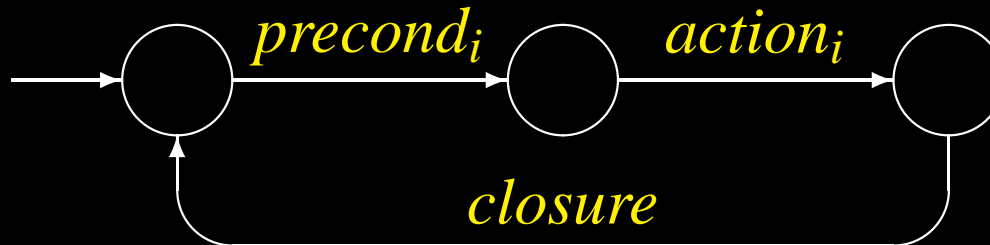
Cognitive Errors

Attention

History of Formal HCI

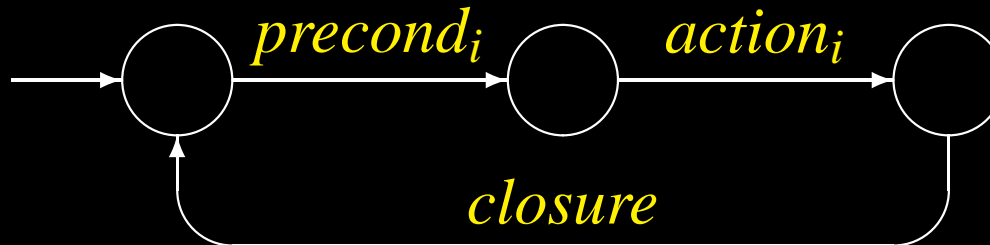
Goals, Actions, Closure

Goal action

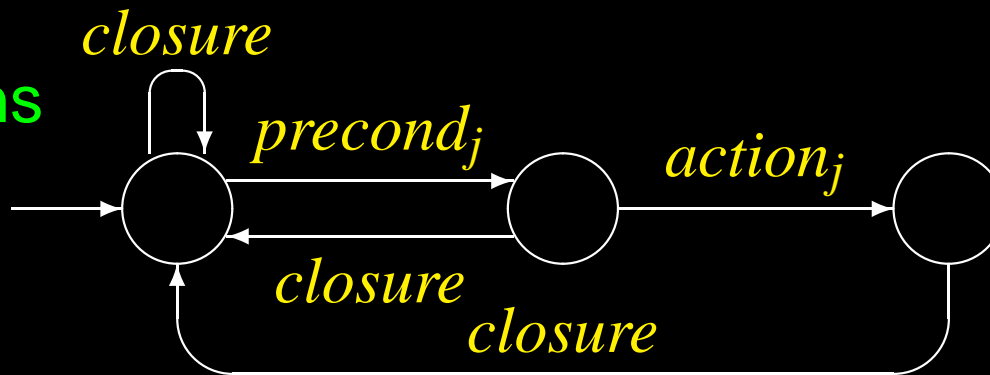


Goals, Actions, Closure

Goal action

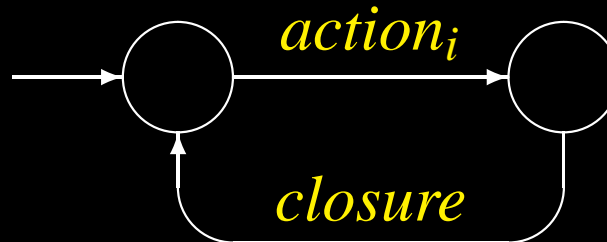


Other actions

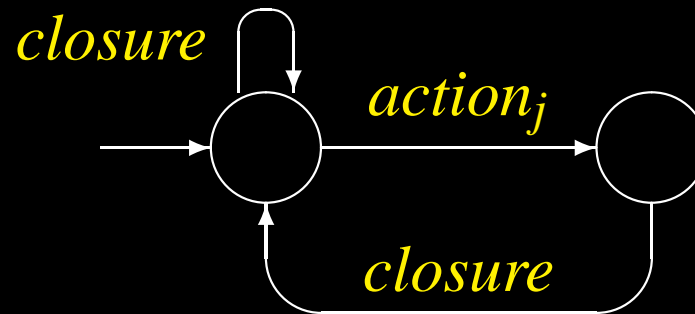


Goals, Actions, Closure

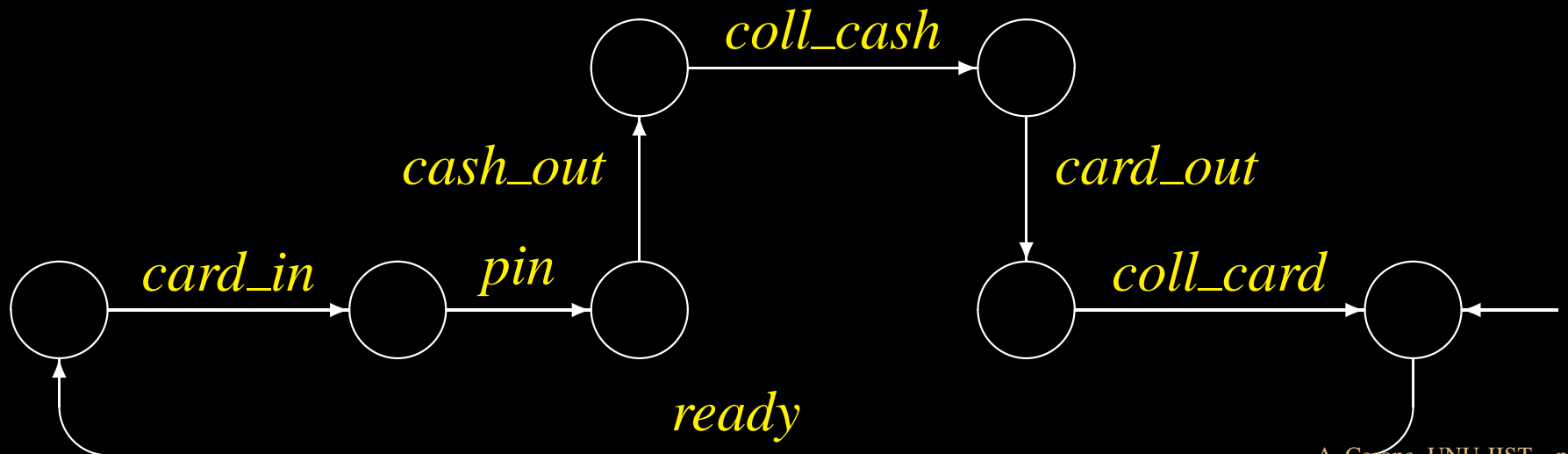
Goal action



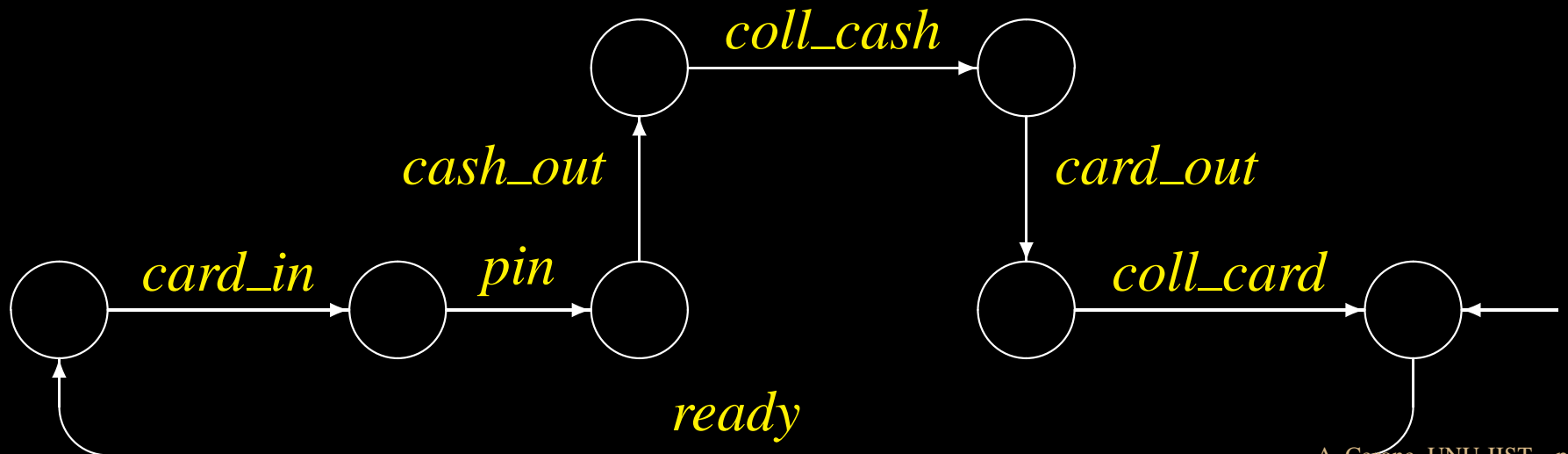
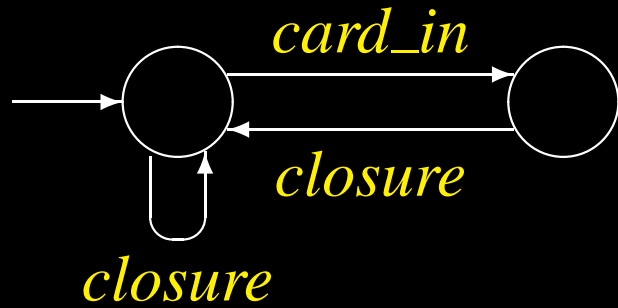
Other actions



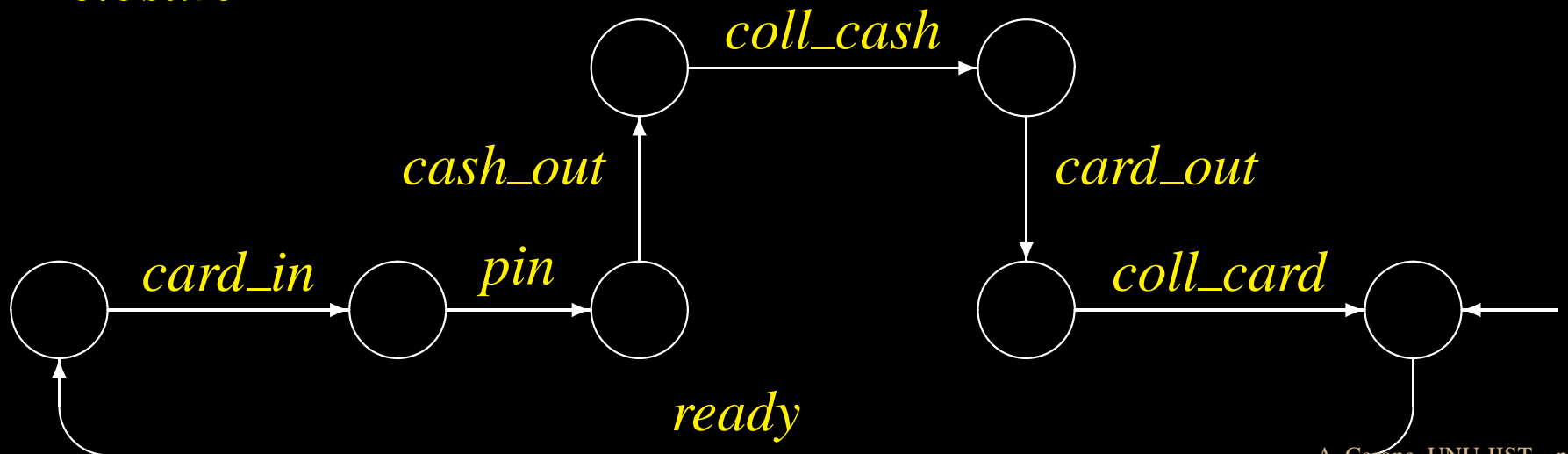
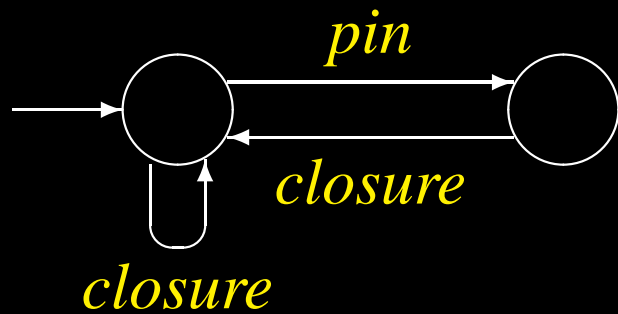
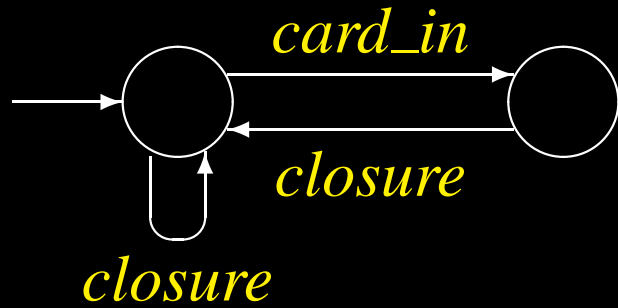
ATM: User



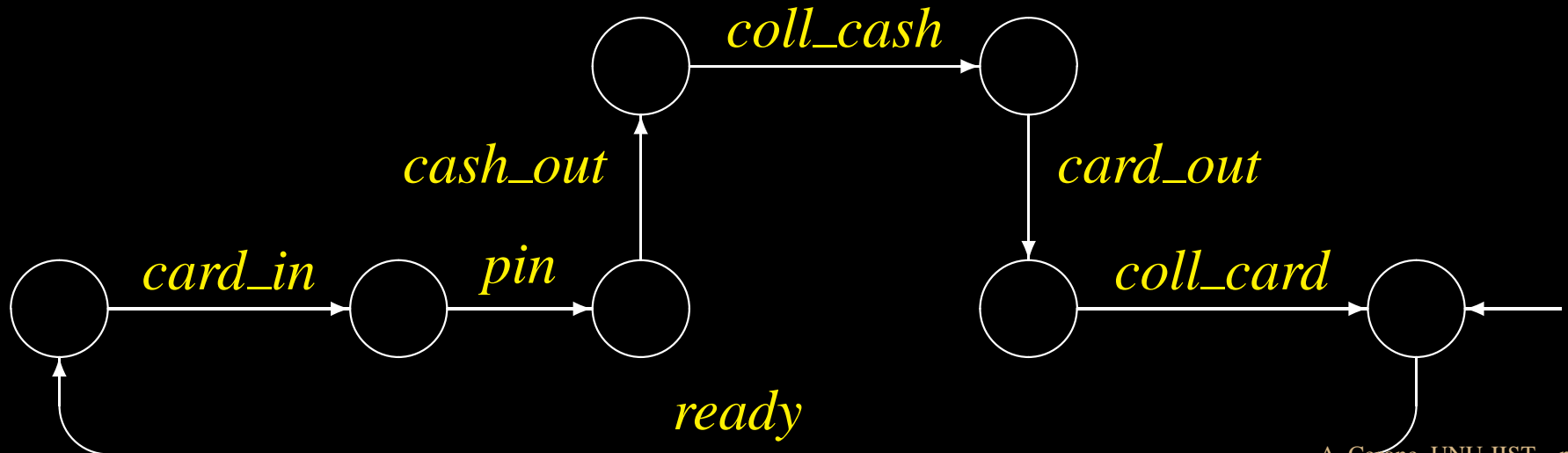
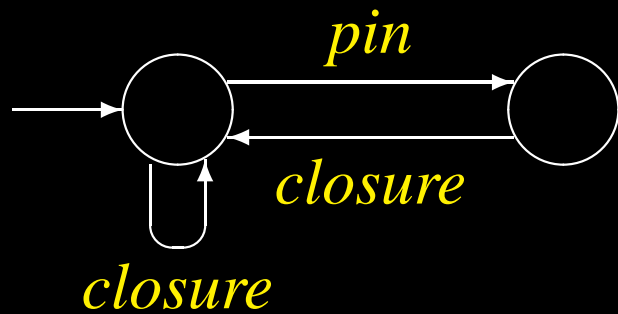
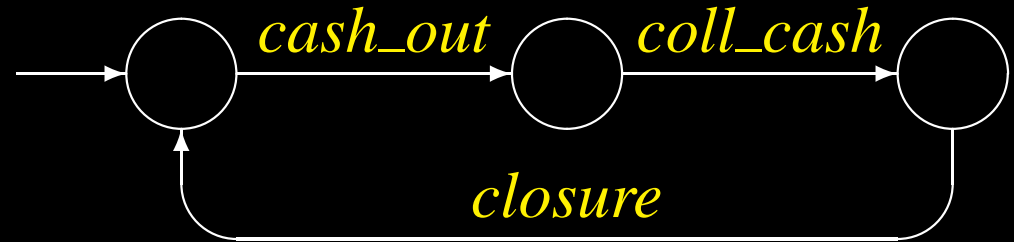
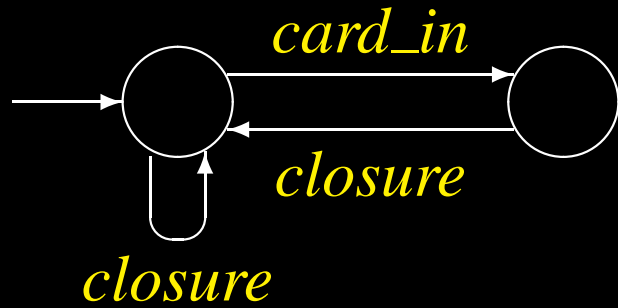
ATM: User



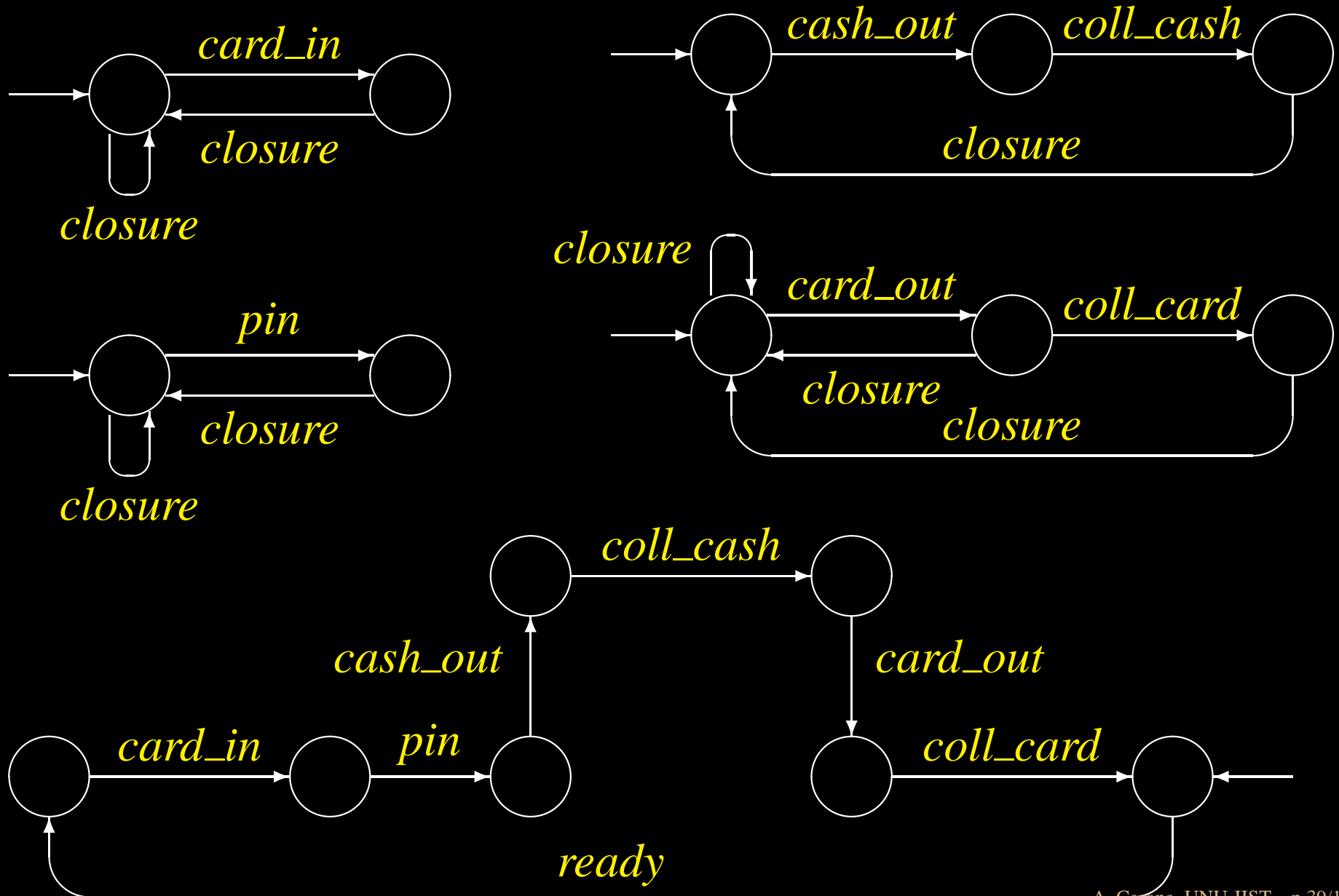
ATM: User



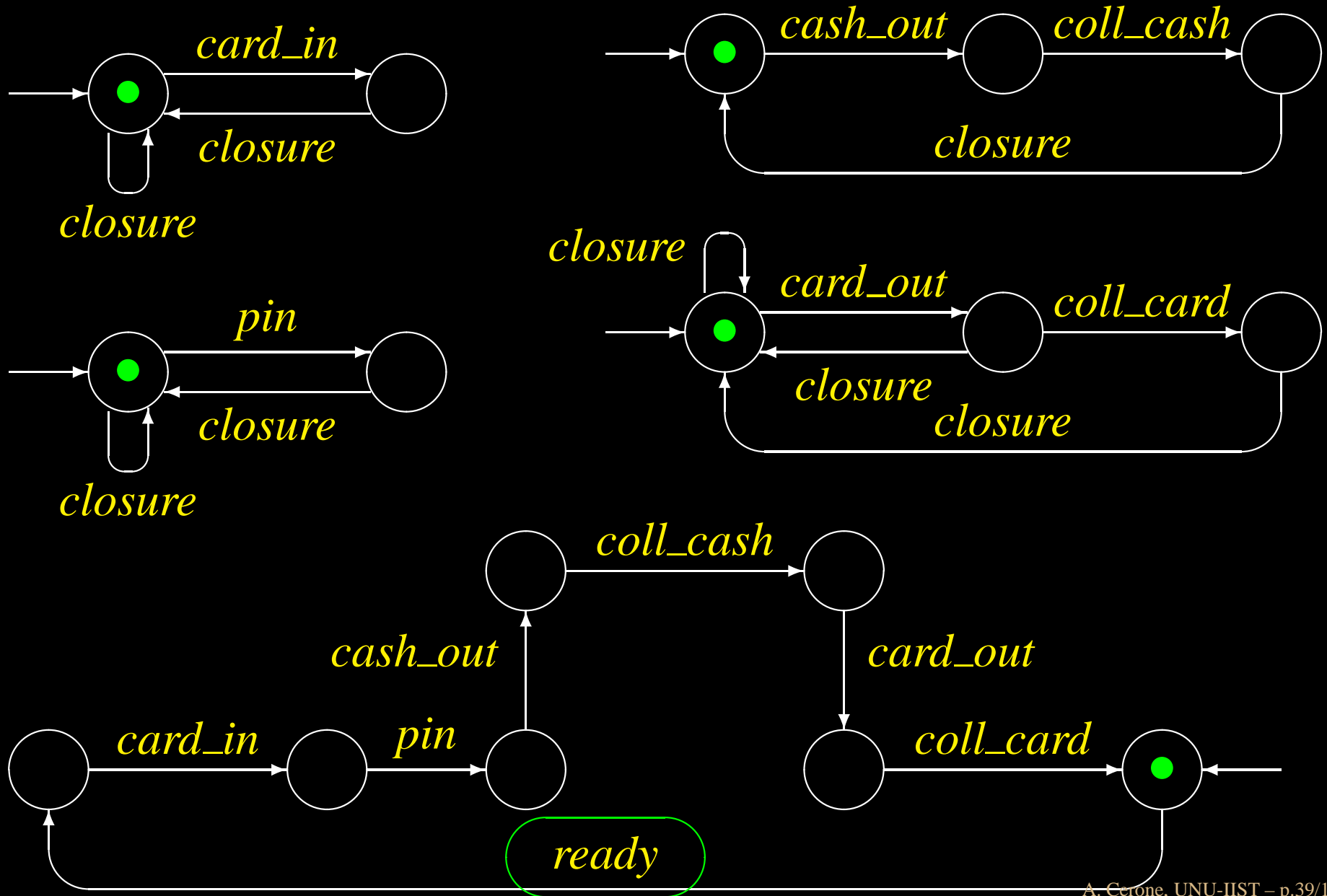
ATM: User



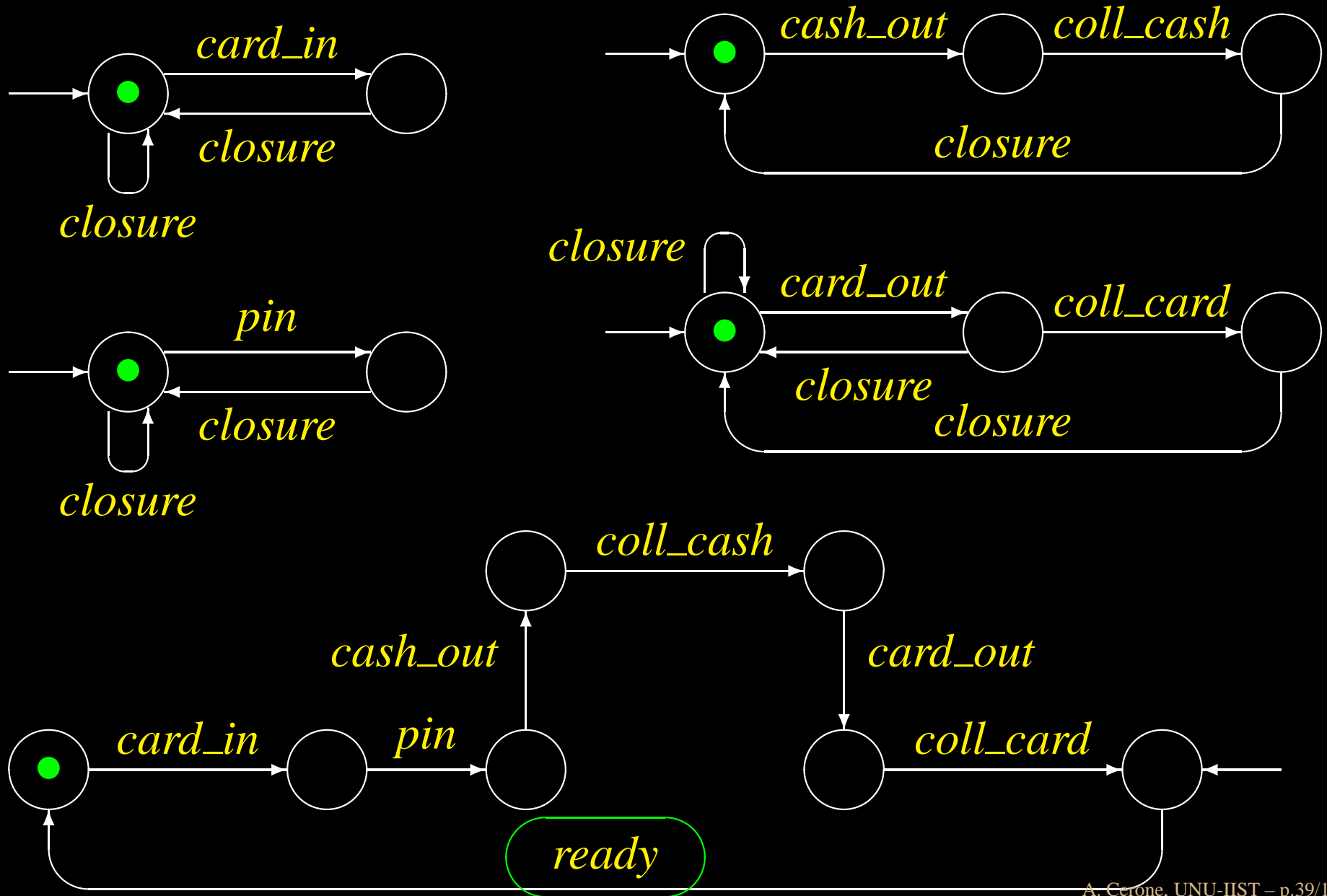
ATM: User



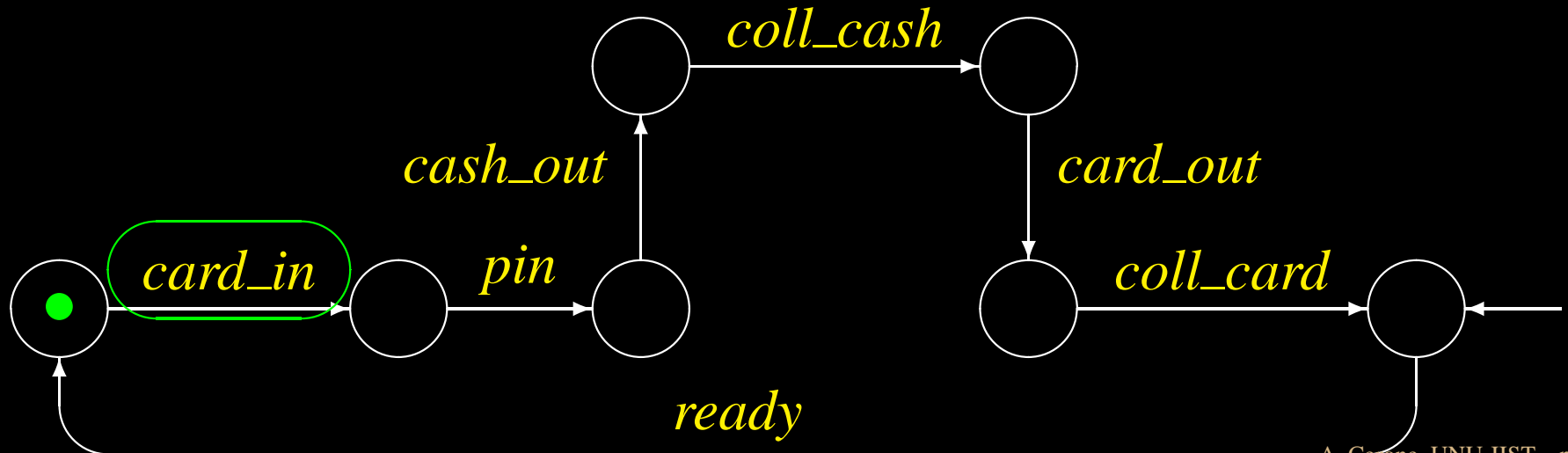
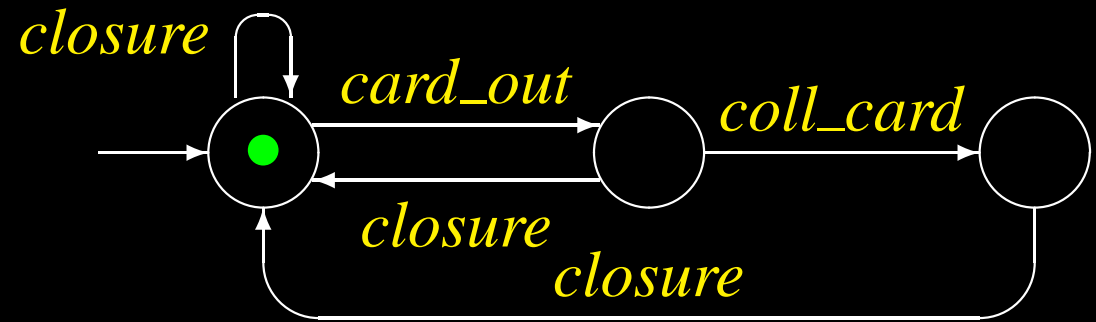
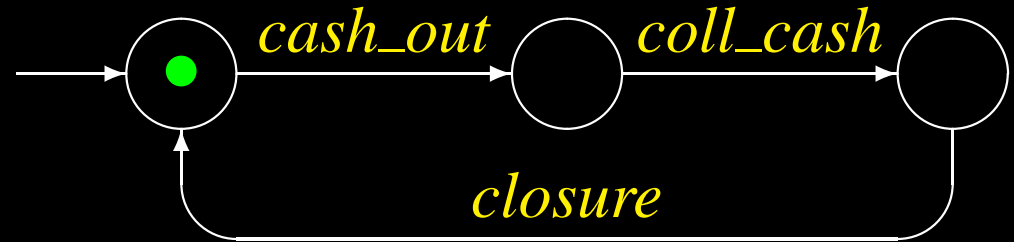
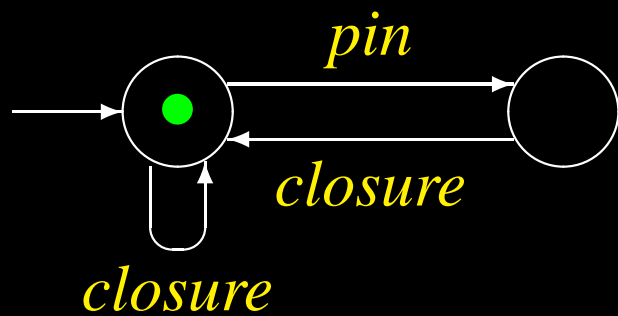
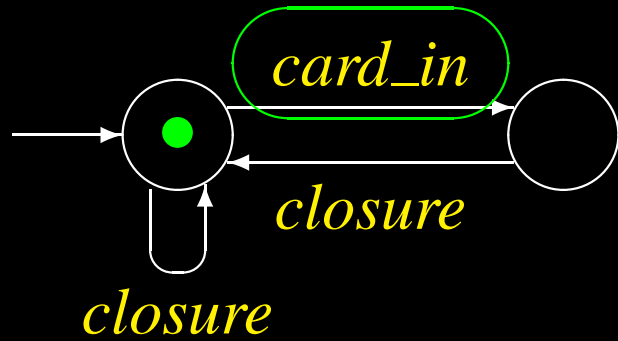
ATM: User



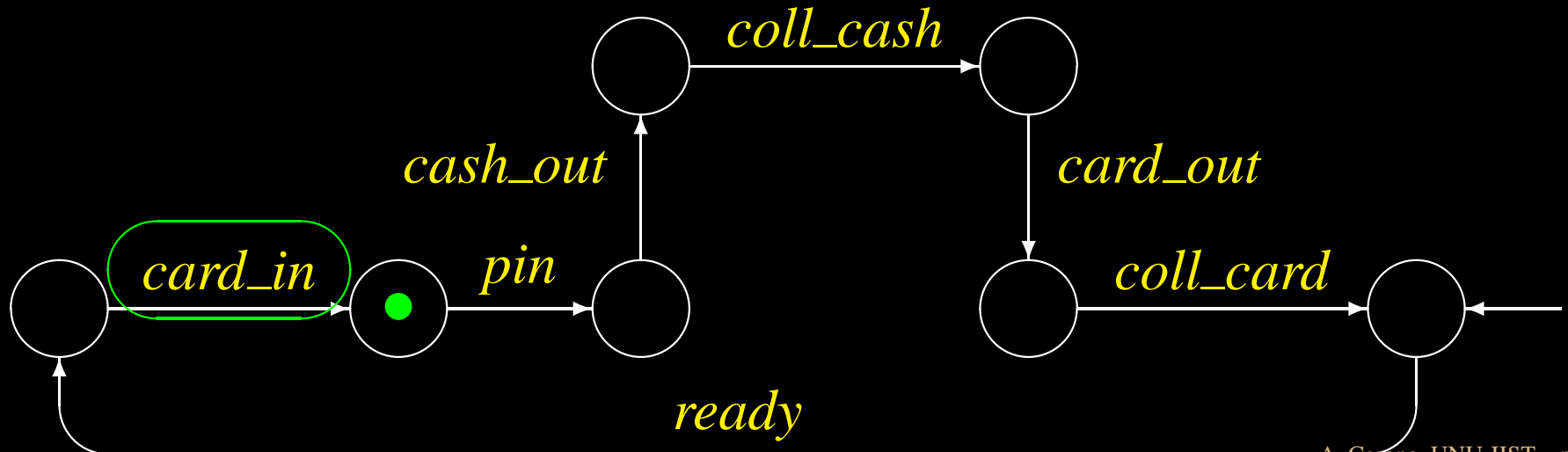
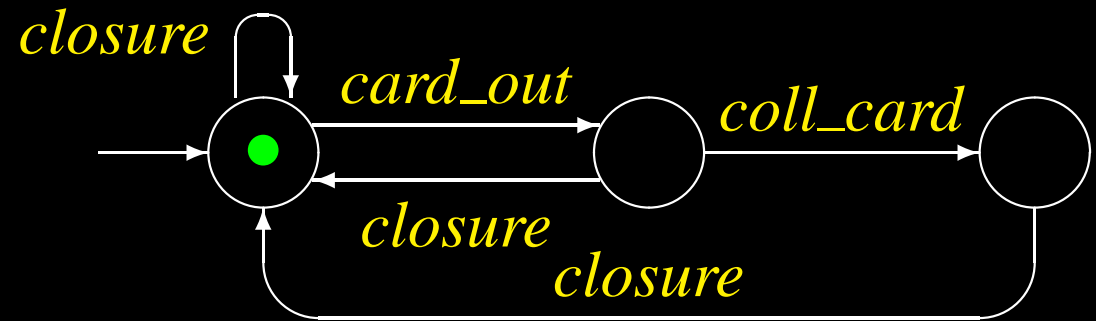
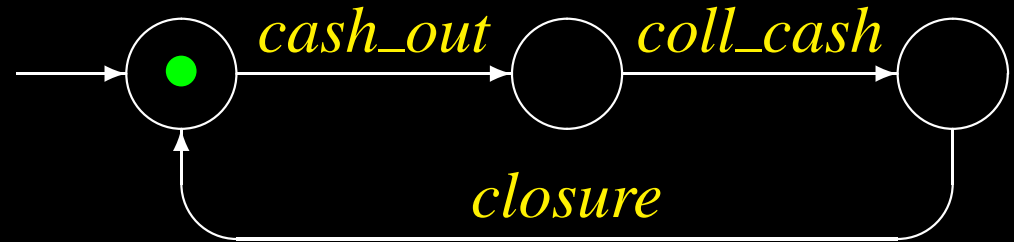
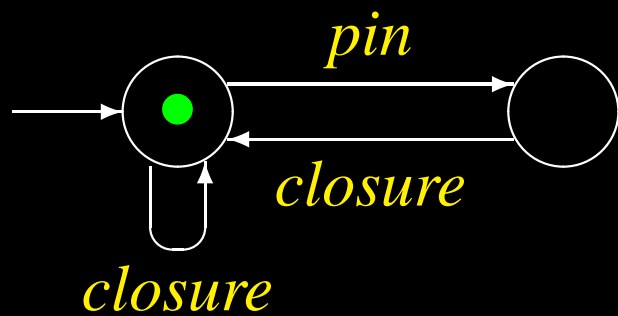
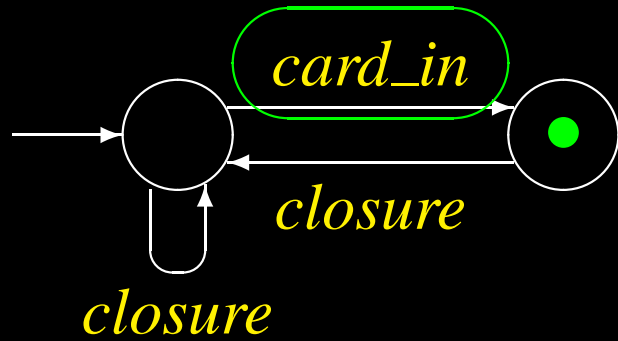
ATM: User



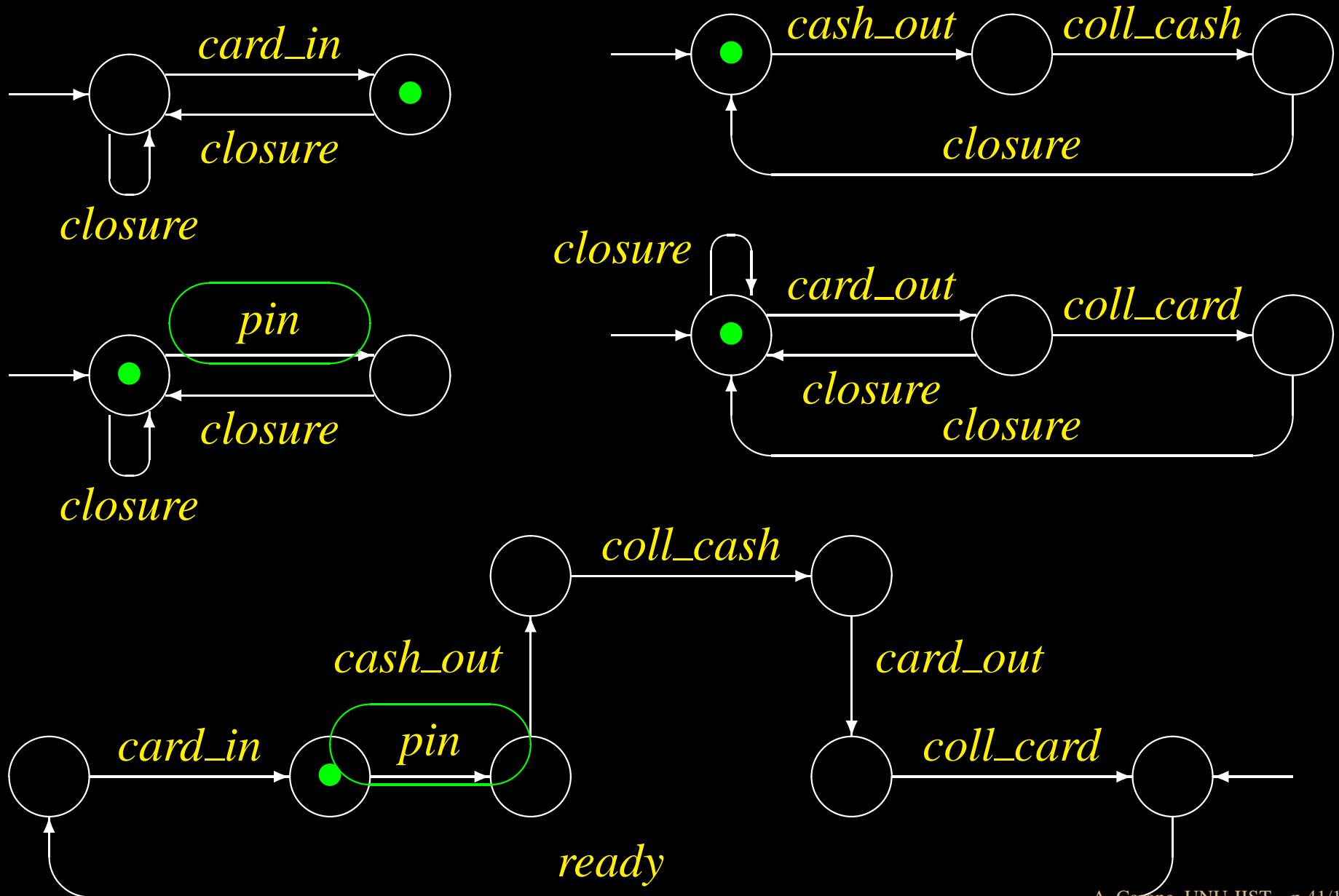
ATM: Interaction 1



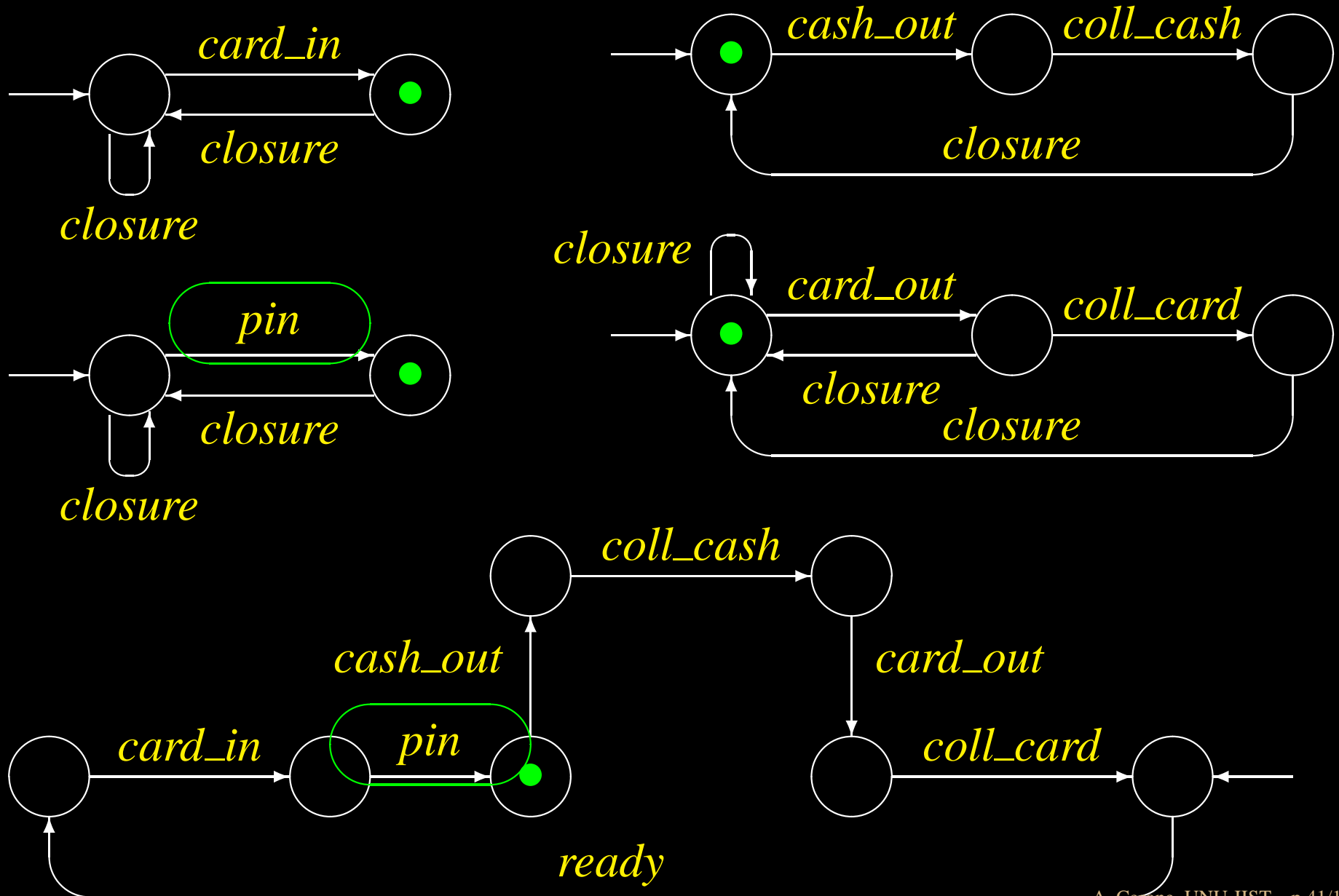
ATM: Interaction 1



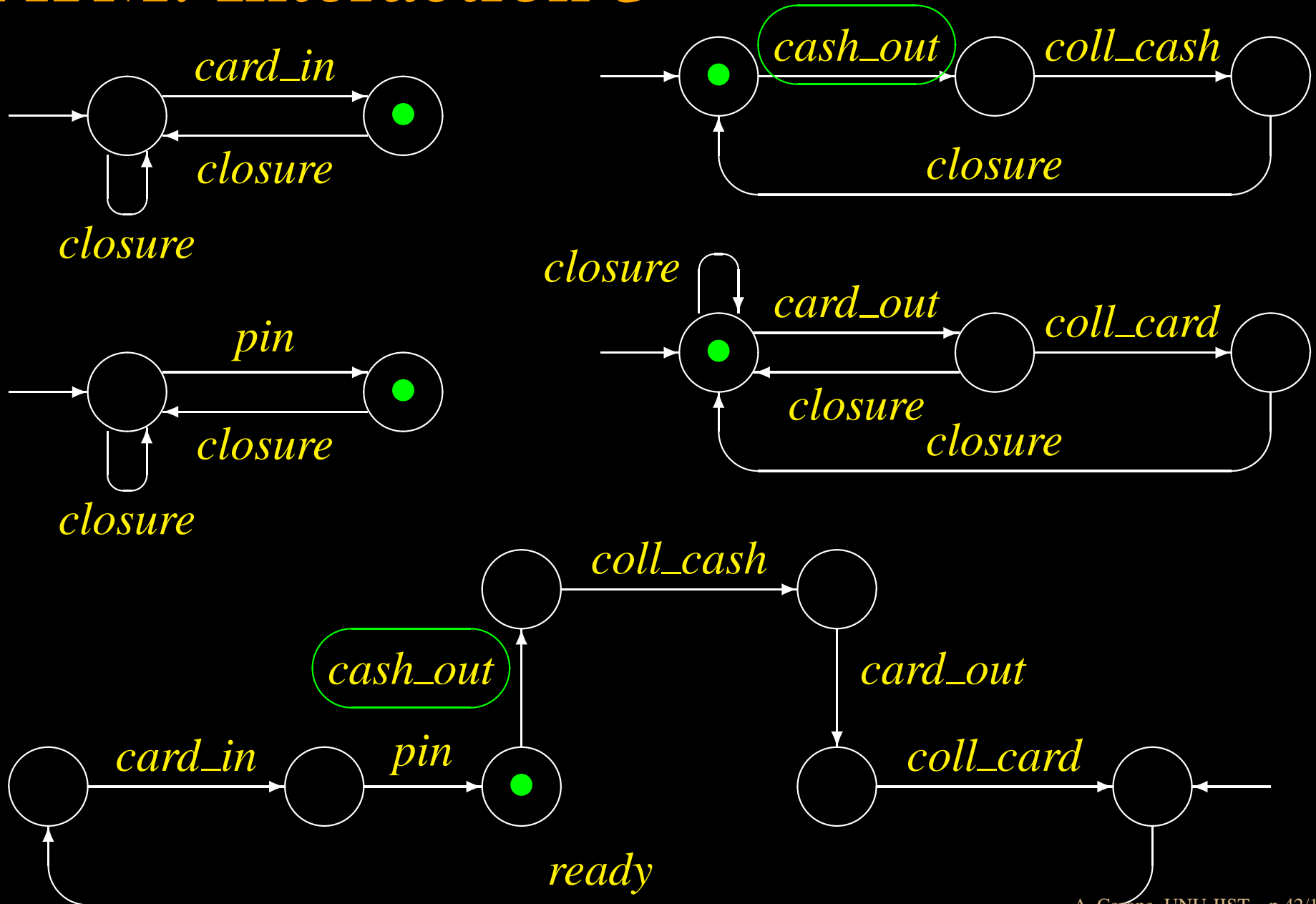
ATM: Interaction 2



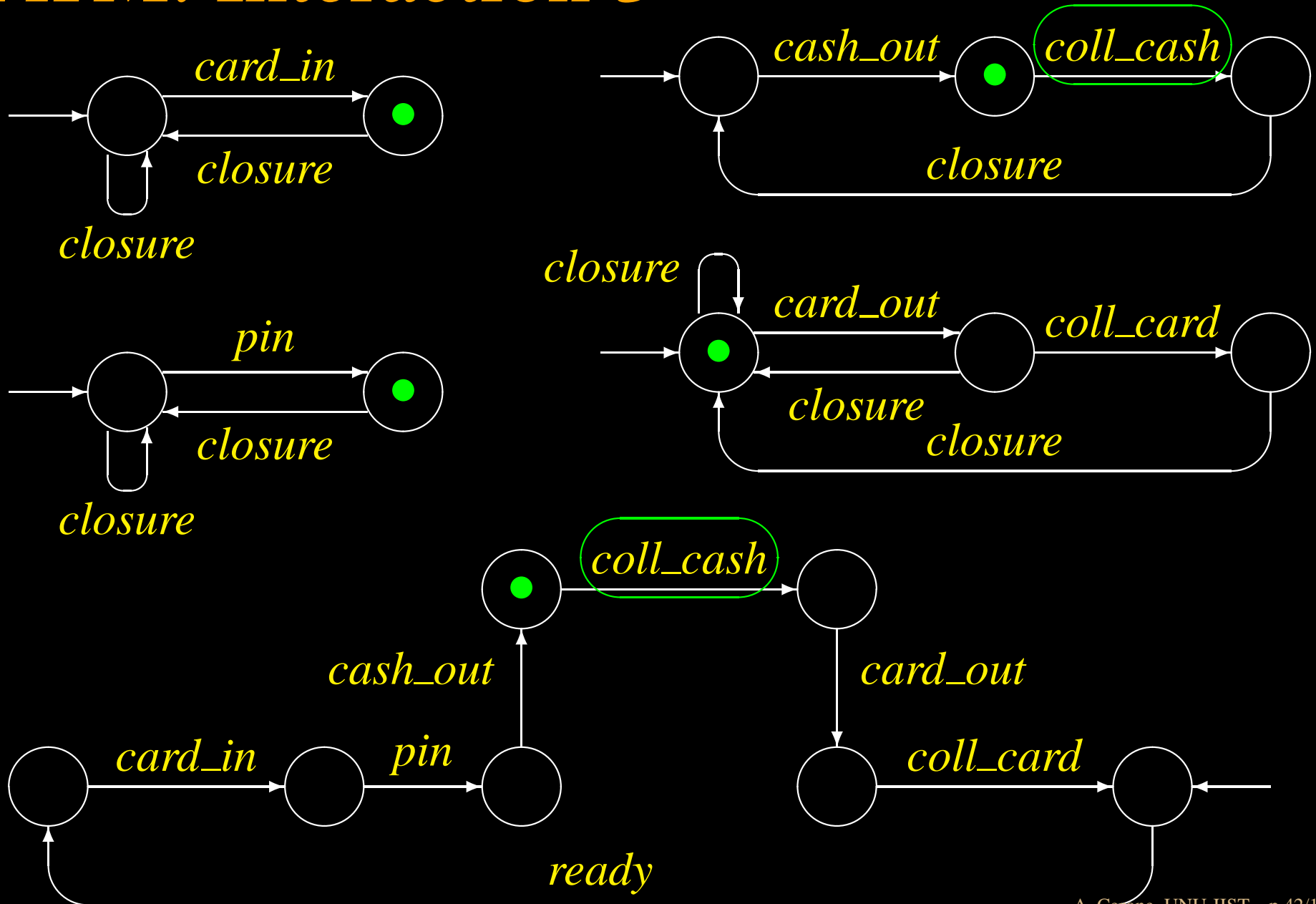
ATM: Interaction 2



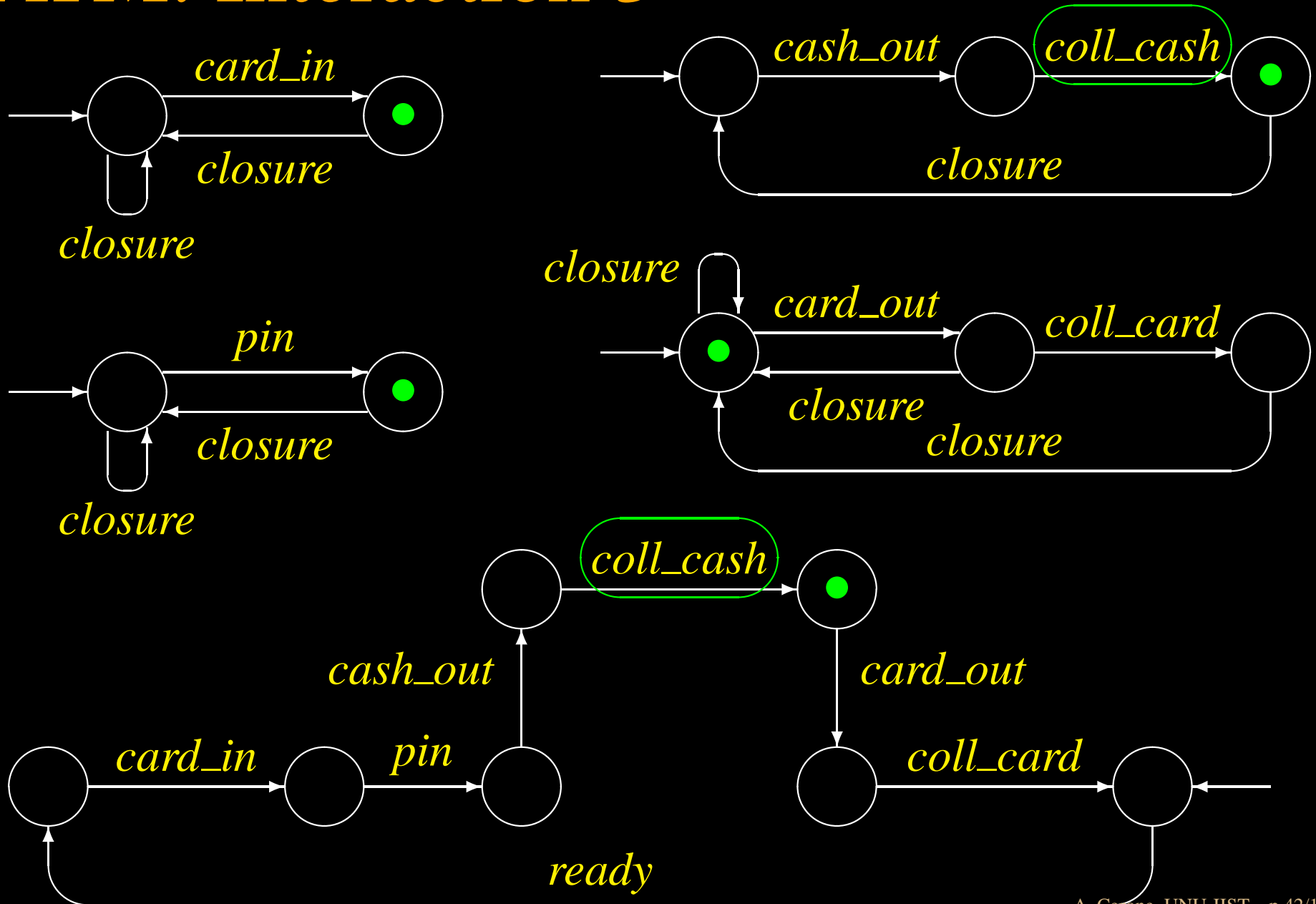
ATM: Interaction 3



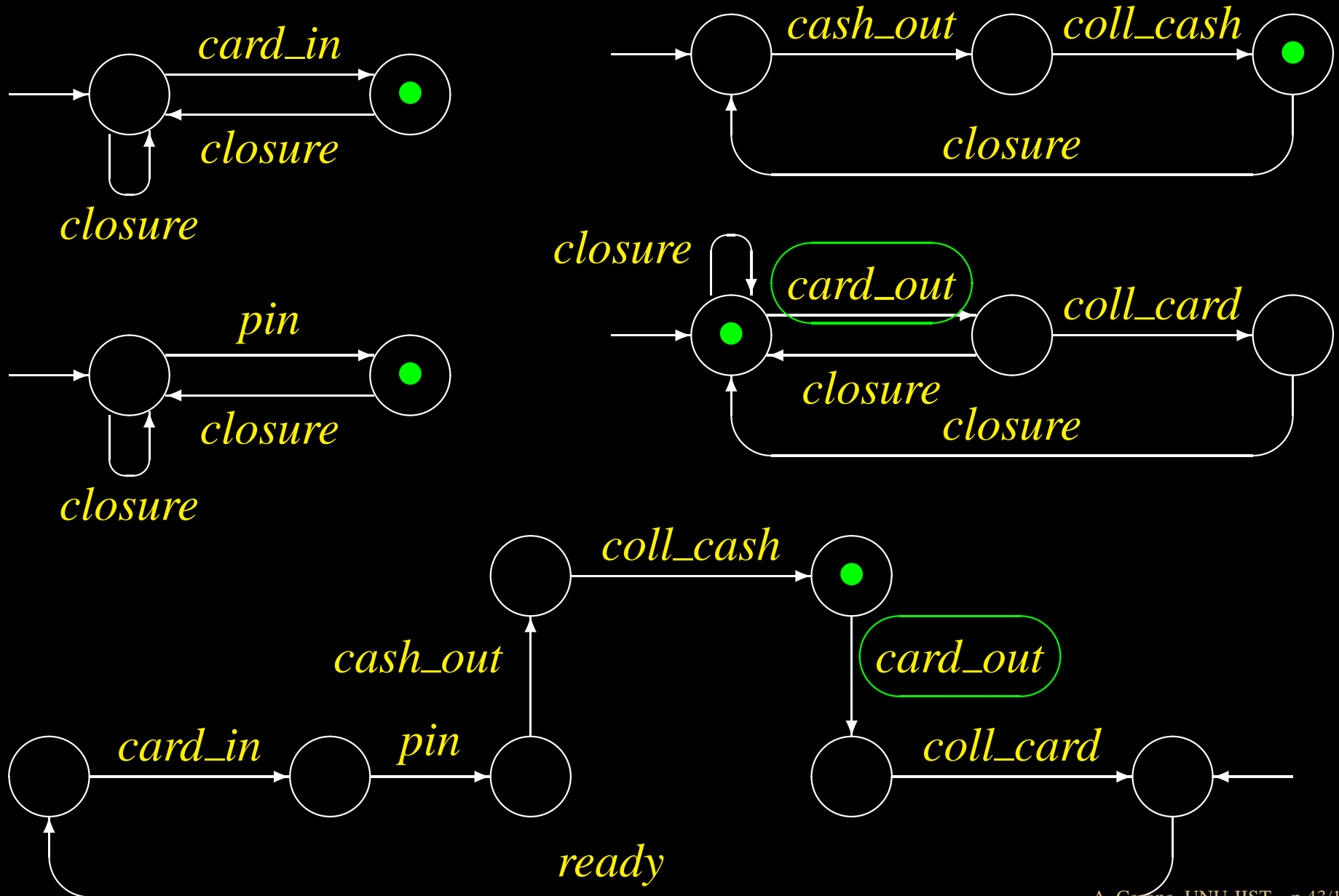
ATM: Interaction 3



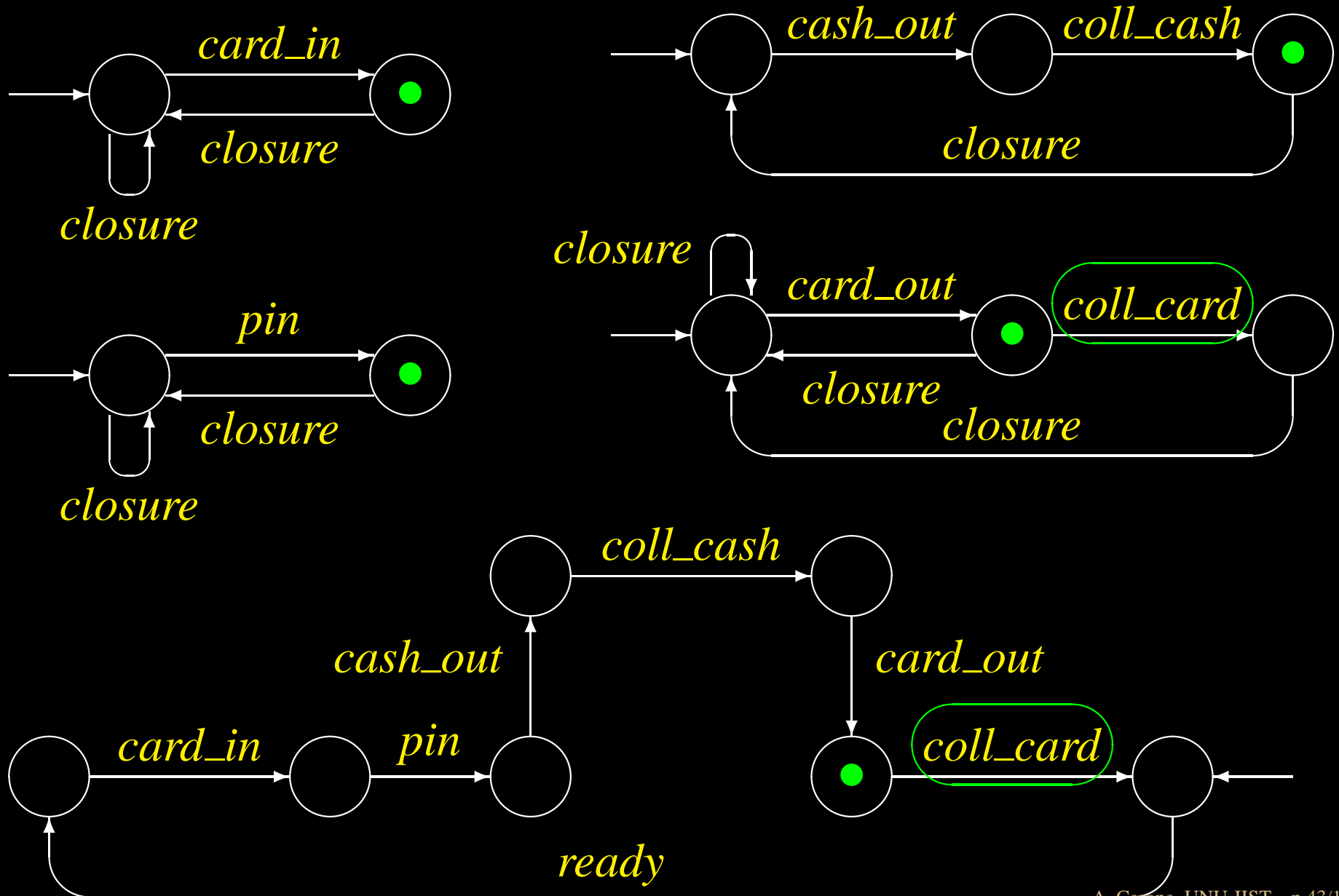
ATM: Interaction 3



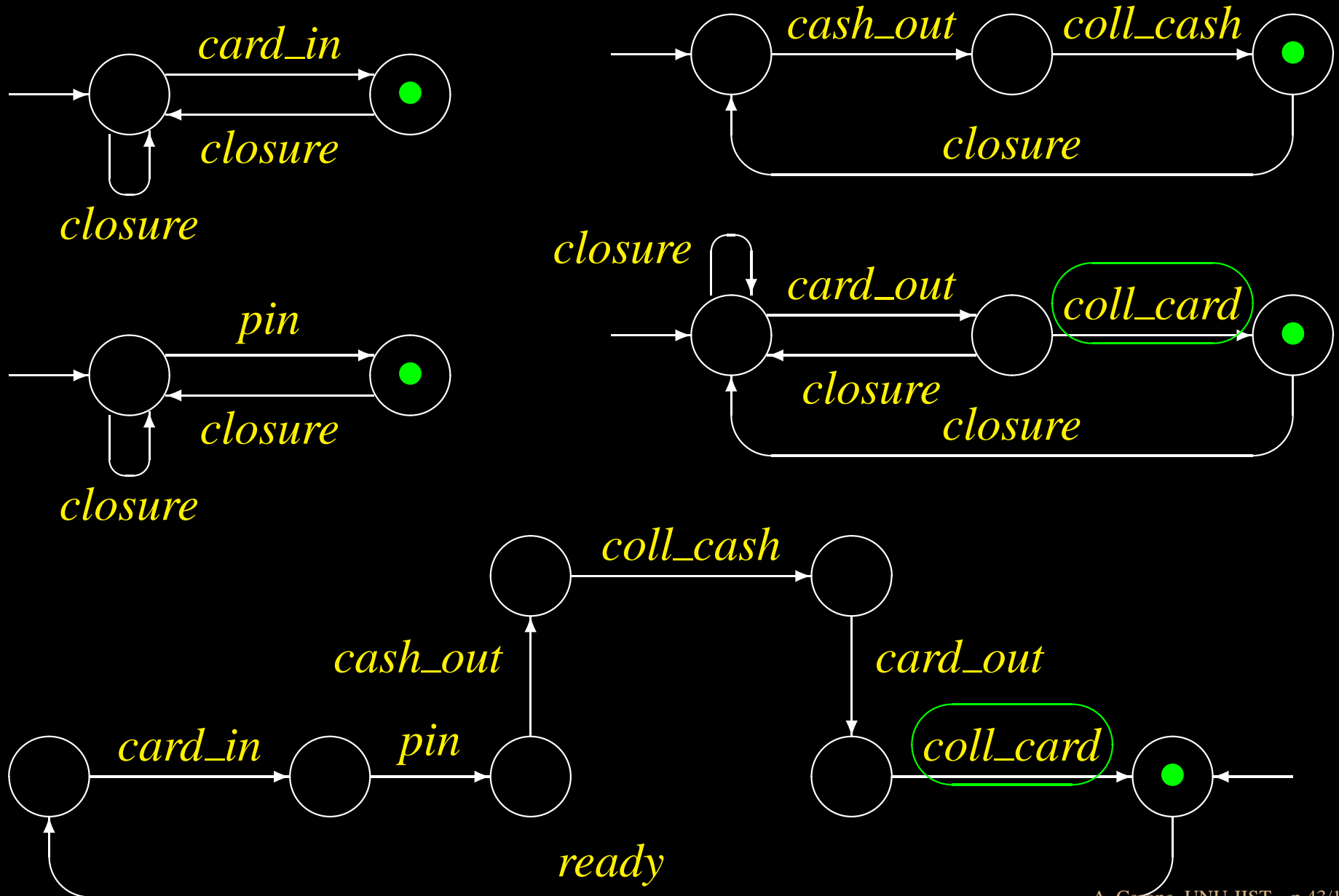
ATM: Interaction 4



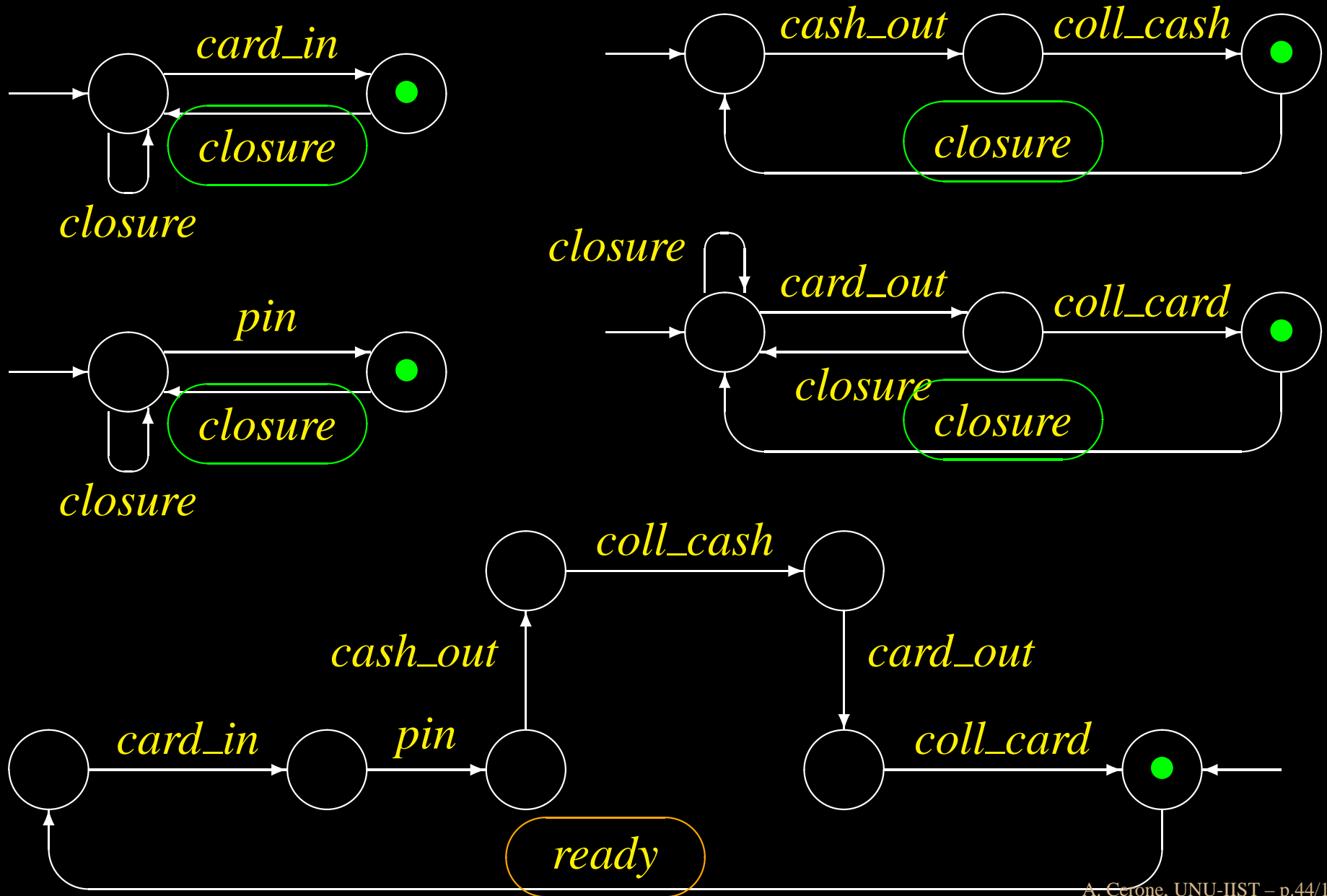
ATM: Interaction 4



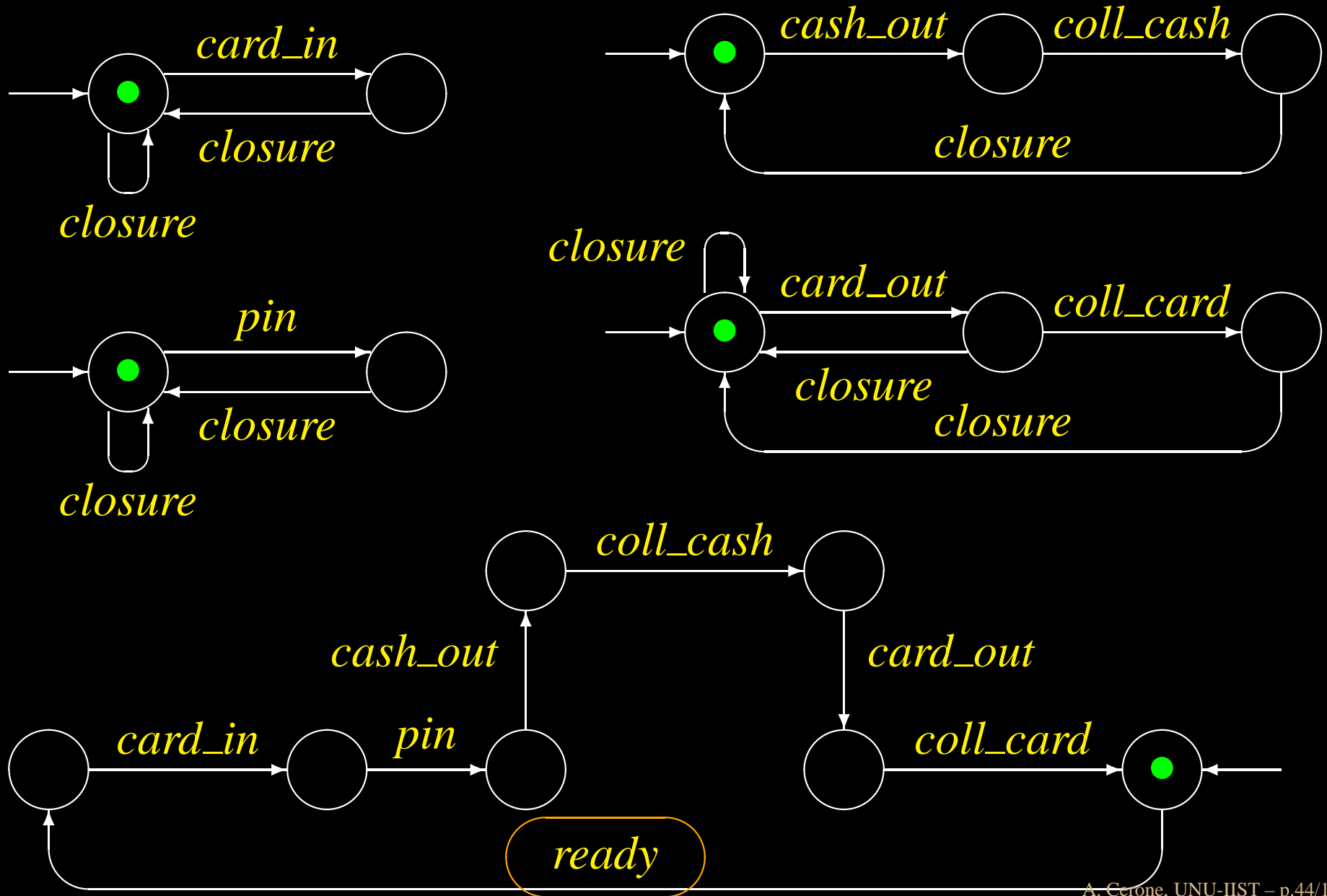
ATM: Interaction 4



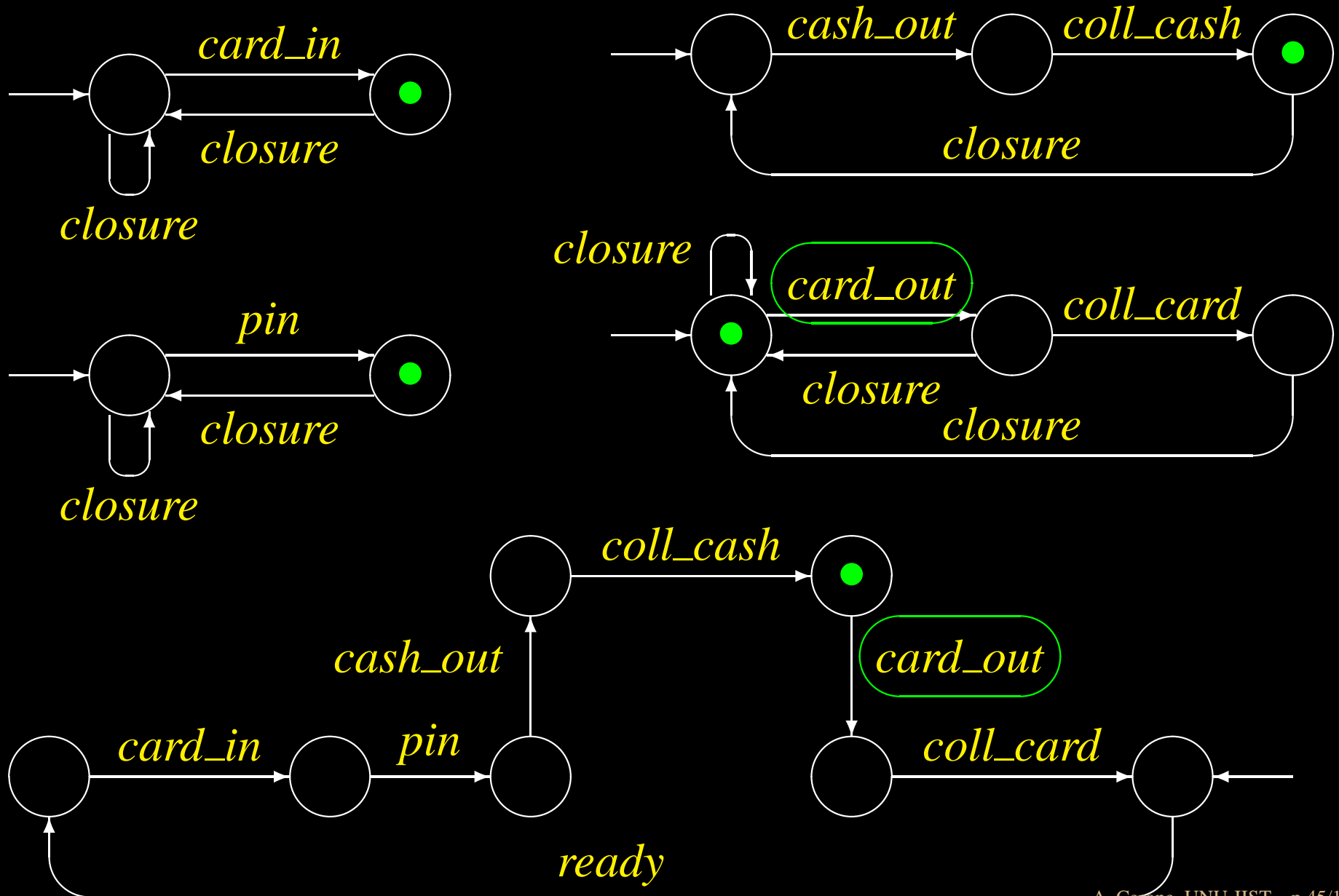
ATM: Closure



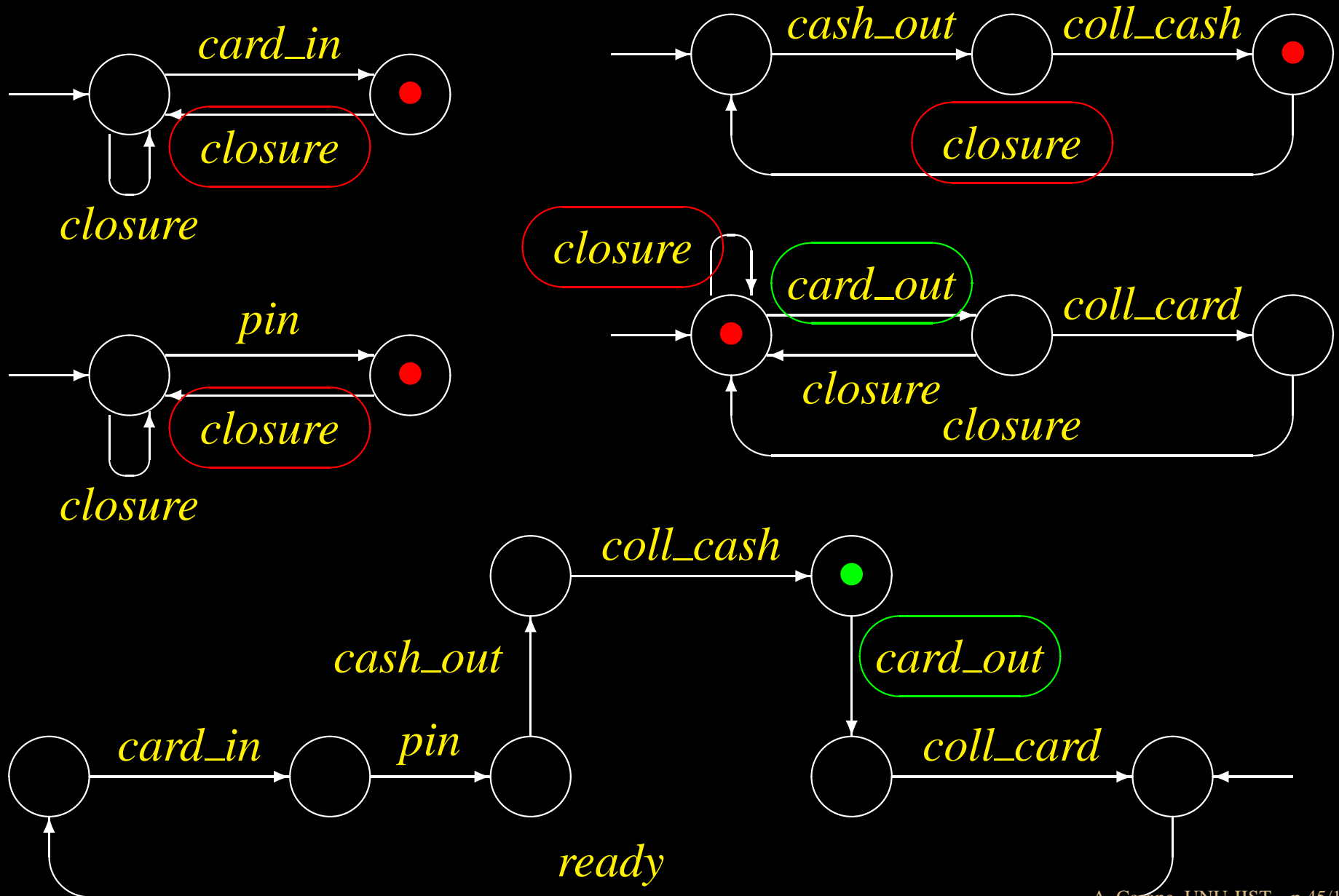
ATM: Closure



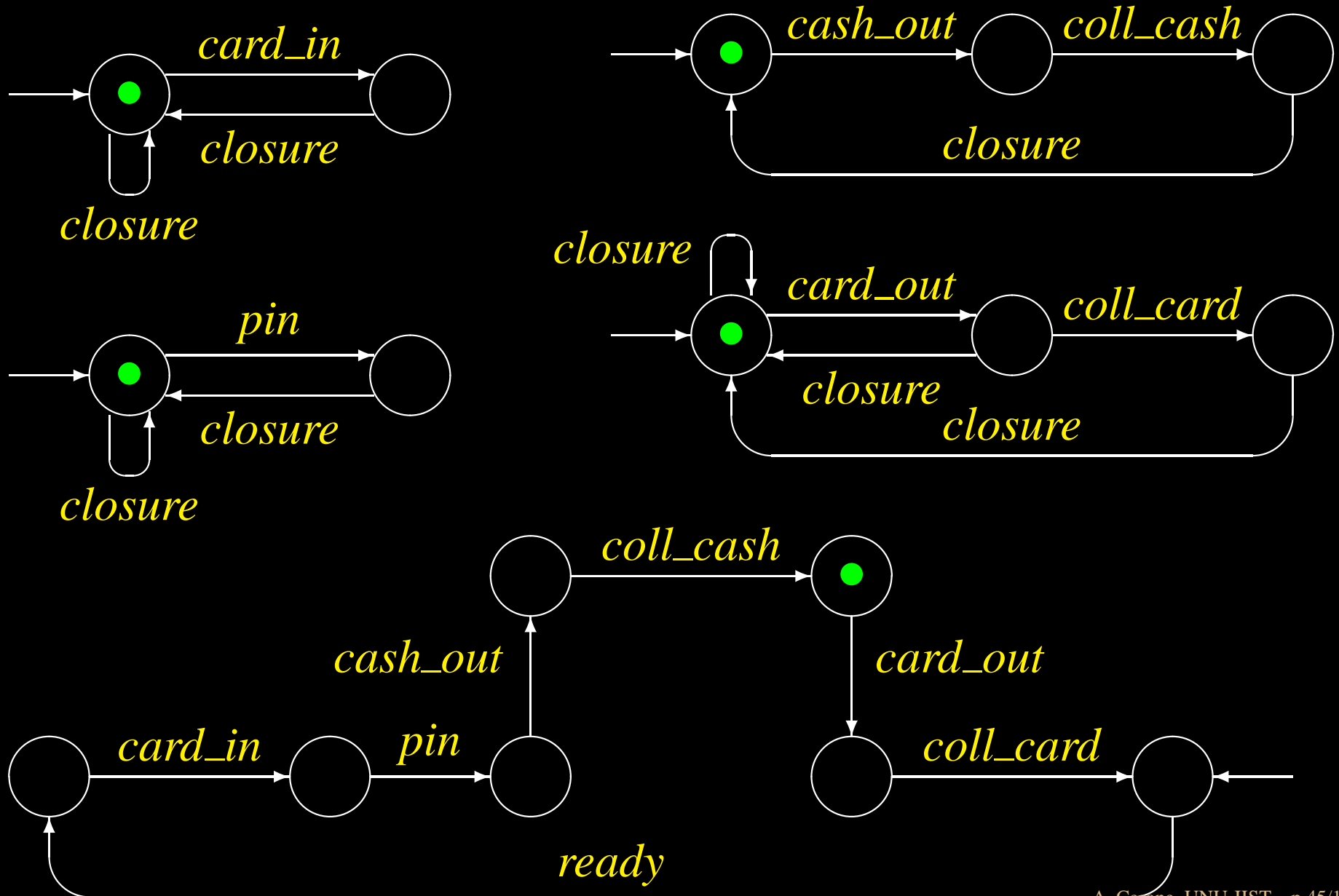
ATM: Post-completion Error



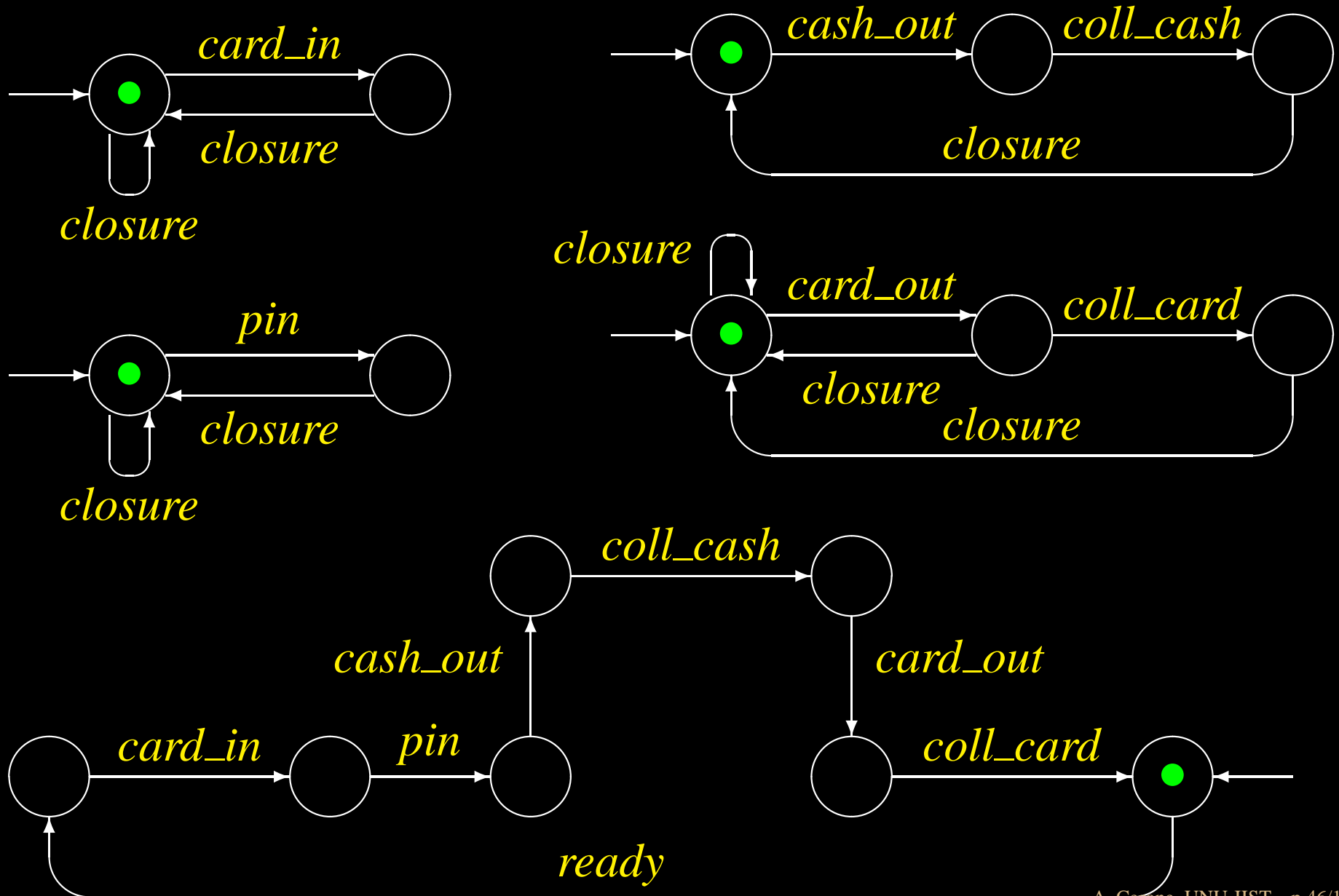
ATM: Post-completion Error



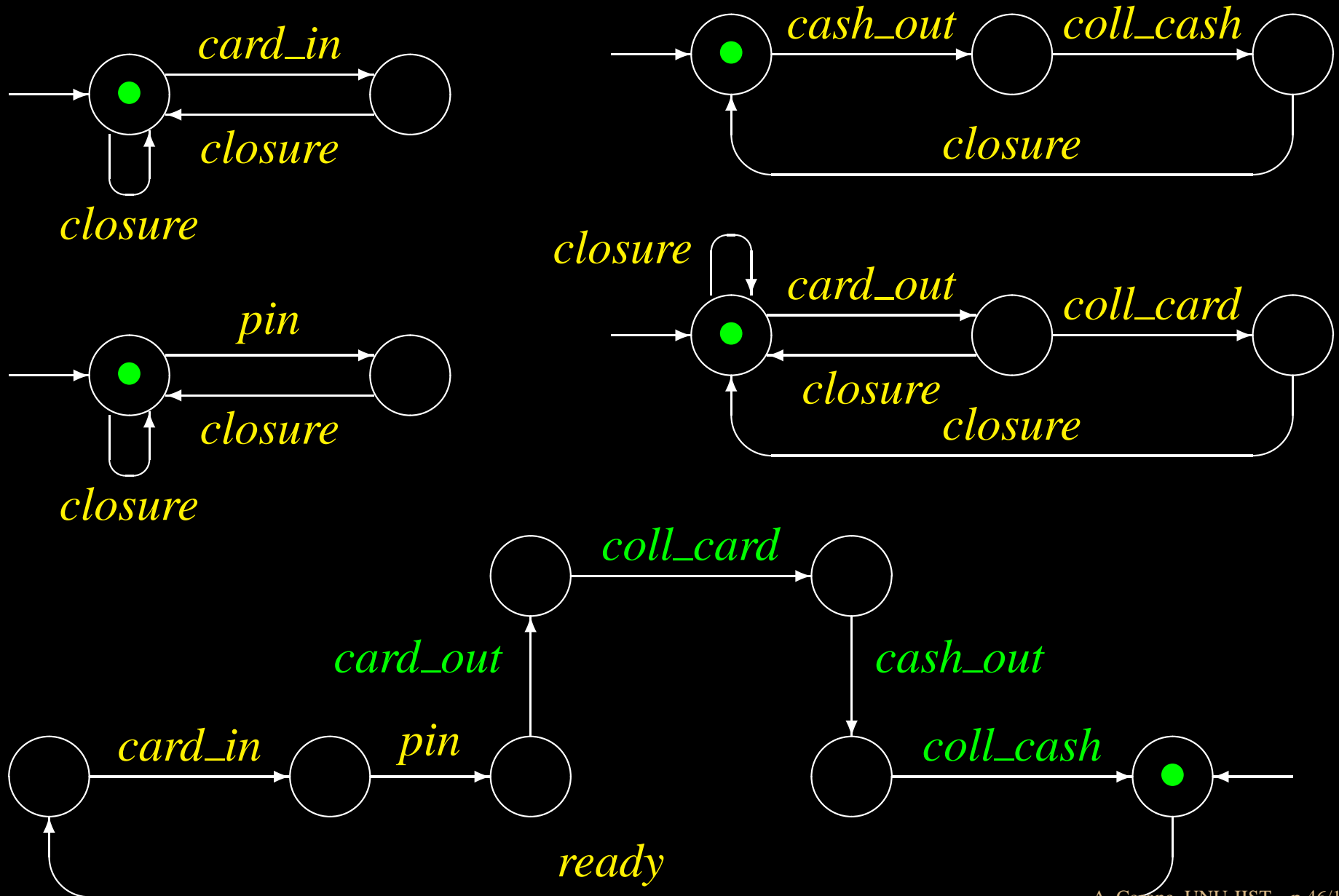
ATM: Post-completion Error



ATM: Correct Machine



ATM: Correct Machine



Cognitive Errors

- Post-completion

Cognitive Errors

- Post-completion
- Impatience

Cognitive Errors

- Post-completion
- Impatience
- Habituation

Cognitive Errors

- Post-completion
- Impatience
- Habituation \Rightarrow security violations

Cognitive Errors

- Post-completion
- Impatience
- Habituation \implies security violations

Thomas Anung Basuki, Antonio Cerone, Andreas Griesmayer and Rudolf Schlatte. **Model-Checking User Behaviour Using Interacting Components**. Formal Aspects of Computing, Vol. 21, No. 6, Springer, 2009.

Cognitive Errors

- Post-completion
- Impatience
- Habituation \implies security violations

Thomas Anung Basuki, Antonio Cerone, Andreas Griesmayer and Rudolf Schlatte. **Model-Checking User Behaviour Using Interacting Components**. Formal Aspects of Computing, Vol. 21, No. 6, Springer, 2009.

- Limited Expertise

Cognitive Errors

- Post-completion
- Impatience
- Habituation \implies security violations

Thomas Anung Basuki, Antonio Cerone, Andreas Griesmayer and Rudolf Schlatte. **Model-Checking User Behaviour Using Interacting Components**. Formal Aspects of Computing, Vol. 21, No. 6, Springer, 2009.

- Limited Expertise \implies security violations

Antonio Cerone and Norzima Elbegbayan. **Model-checking Driven Design of Interactive Systems**. ENTCS 183, Elsevier, pages 3–20, 2007.

Habituation: Exercises

1. Define a constraint that makes a user habituated to input the pin before inserting the card

Habitation: Exercises

1. Define a constraint that makes a user habituated to input the pin before inserting the card
2. Define a constraint that **may eventually make** a user habituated to input the pin before inserting the card

Closure: Exercise

How do you define the closure when you have more than one goal?

Model actions and closure for an ATM that allows to choose between

- cash withdrawal, and
- statements printing

Attention

- selective attention
(sensory memories \implies short-tem memory)

Attention

- selective attention
(sensory memories \implies short-term memory)
- attention **versus** automaticity

Attention

- selective attention
(sensory memories \implies short-term memory)
- attention **versus** automaticity
- Models of Attention
 - Norman and Shallice's Model

Attention

- selective attention
(sensory memories \implies short-term memory)
- attention **versus** automaticity
- Models of Attention
 - Norman and Shallice's Model
 - most responses: fairly automatic control

Attention

- selective attention
(sensory memories \implies short-term memory)
- attention versus automaticity
- Models of Attention
 - Norman and Shallice's Model
 - most responses: fairly automatic control
 - routine of responses

Attention

- selective attention
(sensory memories \implies short-term memory)
- attention **versus** automaticity
- Models of Attention
 - Norman and Shallice's Model
 - most responses: fairly automatic control
 - routine of responses
 - clash between routine activities
 \implies contention scheduling

Attention

- selective attention
(sensory memories \implies short-term memory)
- attention **versus** automaticity
- Models of Attention
 - Norman and Shallice's Model
 - most responses: fairly automatic control
 - routine of responses
 - clash between routine activities
 \implies contention scheduling
 - routine activities **inappropriate**
 \implies attention

Attention

- selective attention
(sensory memories \implies short-term memory)
- attention **versus** automaticity
- Models of Attention
 - Norman and Shallice's Model
 - most responses: fairly automatic control
 - routine of responses
 - clash between routine activities
 \implies contention scheduling
 - routine activities **inappropriate**
 \implies attention activated by
Supervisory Activating System (SAS)

Supervisory Activating System

SAS becomes active whenever the routine selection of operations becomes **inappropriate**

Supervisory Activating System

SAS becomes active whenever the routine selection of operations becomes **inappropriate**
⇒ whenever an individual encounters:

- **required decision**

Supervisory Activating System

SAS becomes active whenever the routine selection of operations becomes **inappropriate**
⇒ whenever an individual encounters:

- required decision
- expectation failure

Supervisory Activating System

SAS becomes active whenever the routine selection of operations becomes **inappropriate**
⇒ whenever an individual encounters:

- required decision
- expectation failure
assessed as
 - danger

Supervisory Activating System

SAS becomes active whenever the routine selection of operations becomes **inappropriate**
⇒ whenever an individual encounters:

- required decision
- expectation failure
assessed as
 - danger
 - novelty

Supervisory Activating System

SAS becomes active whenever the routine selection of operations becomes **inappropriate**
⇒ whenever an individual encounters:

- required decision
- expectation failure
assessed as
 - danger
 - novelty

based on experience / mental model

Supervisory Activating System

SAS becomes active whenever the routine selection of operations becomes **inappropriate**
⇒ whenever an individual encounters:

- required decision
- expectation failure
assessed as
 - danger
 - novelty

based on experience / mental model

- emotion

Supervisory Activating System

SAS becomes active whenever the routine selection of operations becomes **inappropriate**
⇒ whenever an individual encounters:

- required decision
- expectation failure
assessed as

- danger
- novelty

based on experience / mental model

- emotion
 - temptation, anger, ...

SAS activation in ATM

- required decision

SAS activation in ATM

- required decision

selections: kind of transaction, print balance

SAS activation in ATM

- required decision
selections: kind of transaction, print balance
- danger

SAS activation in ATM

- required decision
selections: kind of transaction, print balance
- danger
card returned unexpectedly

SAS activation in ATM

- required decision
selections: kind of transaction, print balance
- danger
card returned unexpectedly
- novelty

SAS activation in ATM

- required decision
selections: kind of transaction, print balance
- danger
card returned unexpectedly
- novelty
keyboard on the screen,
cash given at earlier stage

SAS activation in ATM

- required decision
selections: kind of transaction, print balance
- danger
card returned unexpectedly
- novelty
keyboard on the screen,
cash given at earlier stage
- temptation

SAS activation in ATM

- **required decision**
selections: kind of transaction, print balance
- **danger**
card returned unexpectedly
- **novelty**
keyboard on the screen,
cash given at earlier stage
- **temptation**
message: enter a draw if you withdraw ...

SAS activation in ATM

- required decision
selections: kind of transaction, print balance
- danger
card returned unexpectedly
- novelty
keyboard on the screen,
cash given at earlier stage
- temptation
message: enter a draw if you withdraw ...
- anger

SAS activation in ATM

- **required decision**
selections: kind of transaction, print balance
- **danger**
card returned unexpectedly
- **novelty**
keyboard on the screen,
cash given at earlier stage
- **temptation**
message: enter a draw if you withdraw ...
- **anger**
message: no cash available

SAS activation in ATM (cont)

- required decision
 \Leftarrow choice operator

SAS activation in ATM (cont)

- required decision
 \Leftarrow choice operator
- danger

SAS activation in ATM (cont)

- required decision
 \Leftarrow choice operator
- danger
 \Leftarrow danger response = leave the interaction

SAS activation in ATM (cont)

- required decision
 \Leftarrow choice operator
- danger
 \Leftarrow danger response = leave the interaction
- novelty

SAS activation in ATM (cont)

- required decision
 \Leftarrow choice operator
- danger
 \Leftarrow danger response = leave the interaction
- novelty
 \Leftarrow depends on the specific situation

Danger Response in ATM

```
proc Danger =  
  danger -> leave_int -> Danger  
[] closure -> leave_int -> Danger
```

Danger Response in ATM

```
proc Danger =  
  danger -> leave_int -> Danger  
  [] closure -> leave_int -> Danger  
  [] card_in Danger [] ...
```

Danger Response in ATM

```
proc Danger =  
  danger -> leave_int -> Danger  
  [] closure -> leave_int -> Danger  
  [] card_in Danger [] ...
```

The user will leave the interaction only in case of

- **danger**: user gives up achieving the goal
- **closure**: user has achieved the goal

Danger Response in ATM

```
proc Danger =  
  danger -> leave_int -> Danger  
  [] closure -> leave_int -> Danger  
  [] card_in Danger [] ...
```

The user will leave the interaction only in case of

- **danger**: user gives up achieving the goal
- **closure**: user has achieved the goal

We need to introduce a new action **leave_int** in the user model

Extended User Model — 1

Goal: collect cash

```
proc CollCashStart =  
    start_int -> CollCashToDo
```

Extended User Model — 1

Goal: collect cash

```
proc CollCashStart =  
    start_int -> CollCashToDo  
    cash_out -> CollCashStart
```

Extended User Model — 1

Goal: collect cash

```
proc CollCashStart =  
    start_int -> CollCashToDo  
    cash_out -> CollCashStart  
  
proc CollCashToDo =  
    leave_int -> CollCashStart  
    [ ] cash_out -> coll_cash  
        -> CollCashDone
```

Extended User Model — 1

Goal: collect cash

```
proc CollCashStart =  
    start_int -> CollCashToDo  
    cash_out -> CollCashStart
```

```
proc CollCashToDo =  
    leave_int -> CollCashStart  
    [] cash_out -> coll_cash  
    -> CollCashDone
```

```
proc CollCashDone =  
    closure -> leave_int  
    -> CollCashStart
```

Extended User Model — 2

Non-goal Action: collect card

```
proc CollCardStart =  
    start_int -> CollCardToDo  
    card_out -> CollCardStart
```

Extended User Model — 2

Non-goal Action: collect card

```
proc CollCardStart =  
    start_int -> CollCardToDo  
    card_out -> CollCardStart  
  
proc CollCardToDo =  
    leave_int -> CollCardStart  
[] closure -> CollCardToDo  
[] card_out -> coll_card  
    -> CollCardDone
```

Extended User Model — 2

Non-goal Action: collect card

```
proc CollCardStart =  
    start_int -> CollCardToDo  
    card_out -> CollCardStart  
  
proc CollCardToDo =  
    leave_int -> CollCardStart  
    [] closure -> CollCardToDo  
    [] card_out -> coll_card  
        -> CollCardDone  
  
proc CollCardDone =  
    leave-Int -> CollCardStart  
    [] closure -> CollCardToDo
```

Extended User Model — 3

Non-goal Action: insert card

```
proc CardInStart =  
    start_int -> CardToDo
```


Extended User Model — 3

Non-goal Action: insert card

```
proc CardInStart =  
    start_int -> CardToDo  
  
proc CardToDo =  
    leave_int -> CardInStart  
[] closure -> CardToDo  
[] card_in -> CardInDone
```

Extended User Model — 3

Non-goal Action: insert card

```
proc CardInStart =  
    start_int -> CardToDo  
  
proc CardToDo =  
    leave_int -> CardInStart  
    [] closure -> CardToDo  
    [] card_in -> CardInDone  
  
proc CardInDone =  
    leave-Int -> CardInStart  
    [] closure -> CardInToDo
```

Modelling SAS in ATM

- Routine Expectations \implies automaticity

Modelling SAS in ATM

- Routine Expectations \implies automaticity
 - expect card_out
 - expect cash_out

Modelling SAS in ATM

- Routine Expectations \implies automaticity
 - expect card_out
 - expect cash_out
- Expectations Failure activates SAS
 - cash_out when card_out expected
 - card_out when cash_out expected

Modelling SAS in ATM

- Routine Expectations \implies automaticity
 - expect card_out
 - expect cash_out
- Expectations Failure activates SAS
 - cash_out when card_out expected
 - card_out when cash_out expected
- Attention Response
 - assessment (danger or novelty)

Modelling SAS in ATM

- Routine Expectations \implies automaticity
 - expect card_out
 - expect cash_out
- Expectations Failure activates SAS
 - cash_out when card_out expected
 - card_out when cash_out expected
- Attention Response
 - assessment (danger or novelty)
 - action (leave_int or specific)

Modelling SAS in ATM

- Routine Expectations \implies automaticity
 - expect card_out
 - expect cash_out
 - Expectations Failure activates SAS
 - cash_out when card_out expected
 - card_out when cash_out expected
 - Attention Response
 - assessment (danger or novelty)
 - action (leave_int or specific)
- based on experience / mental model

Routine Expectations in ATM

- expect **cash_out** before **card_out**

Routine Expectations in ATM

- expect **cash_out** before **card_out**

```
proc Expectations =  
    pin -> expect_cash_out  
        -> Expectations  
[] coll_cash -> expect_card_out  
        -> Expectations
```

Routine Expectations in ATM

- expect **cash_out** before **card_out**

```
proc Expectations =  
    pin -> expect_cash_out  
        -> Expectations  
    [] coll_cash -> expect_card_out  
        -> Expectations
```

- expect **card_out** before **cash_out**

Routine Expectations in ATM

- expect **cash_out** before **card_out**

```
proc Expectations =  
    pin -> expect_cash_out  
        -> Expectations  
    [] coll_cash -> expect_card_out  
        -> Expectations
```

- expect **card_out** before **cash_out**

```
proc Expectations =  
    pin -> expect_card_out  
        -> Expectations  
    [] coll_card -> expect_cash_out  
        -> Expectations
```

Expectations Failure in ATM

```
proc SAS = start_int -> Activation
    [] card_out -> SAS
    [] csh_out -> SAS
```

Card Expectations Failure

```
proc SAS = start_int -> Activation
    [] card_out -> SAS
    [] csh_out -> SAS
```

```
proc Activation = expect_card_out ->
    ( card_out -> expect_met
      -> Activation
    [] cash_out -> cash_no_card
      -> Activation
    [] leave_int -> SAS )
```

Card Expectations Failure

```
proc SAS = start_int -> Activation
    [] card_out -> SAS
    [] csh_out -> SAS
```

```
proc Activation = expect_card_out ->
    ( card_out -> expect_met
      -> Activation
    [] cash_out -> cash_no_card
      -> Activation
    [] leave_int -> SAS )
[] expect_cash_out -> ...
```

Card Expectations Failure

```
proc SAS = start_int -> Activation
    [] card_out -> SAS
    [] csh_out -> SAS
```

```
proc Activation = expect_card_out ->
    ( card_out -> expect_met
      -> Activation
    [] cash_out -> cash_no_card
      -> Activation
    [] leave_int -> SAS )

[] expect_cash_out -> ...

[] leave_int -> SAS )
```


Cash Expectations Failure

```
proc SAS = start_int -> Activation
    [] card_out -> SAS
    [] csh_out -> SAS
```

```
proc Activation = expect_card_out ->
    ...
```

```
    [] expect_cash_out -> ...
        ( cash_out -> expect_met
          -> Activation
        [] card_out -> card_no_cash
          -> Activation
        [] leave_int -> SAS )
    [] leave_int -> SAS )
```

Interaction with SAS in ATM

```
proc Interaction_with_SAS =  
  ( Interaction  
    || {start_int, card_out,  
        cash_out, leave_int} || SAS )  
    || {closure, leave_int, card_in,  
        pin, coll_card, coll_cash} ||  
  Danger
```

Failure Assessment in ATM

```
proc Assess =
```

```
    card_no_cash -> coll_card
```

```
        -> danger -> Assess           % danger
```

Failure Assessment in ATM

```
proc Assess =  
  card_no_cash -> coll_card  
    -> danger -> Assess          % danger  
  cash_no_card-> coll_cash  
    -> Assess
```

Failure Assessment in ATM

```
proc Assess =  
  card_no_cash -> coll_card  
    -> danger -> Assess          % danger  
  cash_no_card-> coll_cash  
    -> Assess                    % novelty
```

Failure Assessment in ATM

```
proc Assess =  
  card_no_cash -> coll_card  
    -> danger -> Assess          % danger  
  cash_no_card-> coll_cash  
    -> Assess                    % novelty  
  expect_met ->  
    (coll_cash -> Assess  
  [ ] coll_card -> Assess)
```

Failure Assessment in ATM

```
proc Assess =  
  card_no_cash -> coll_card  
    -> danger -> Assess          % danger  
  cash_no_card-> coll_cash  
    -> Assess                    % novelty  
  expect_met ->  
    (coll_cash -> Assess  
  [ ] coll_card -> Assess)
```

based on task knowledge
and maybe experience / mental model

Attention Response in ATM

```
proc Attention_Response =  
  ( Interaction_with_SAS  
    || {pin,expect_cash_out,  
       coll_cash,expect_card_out} ||  
    Expectations )  
  || {expect_met,  
     card_no_cash,cash_no_card,  
     coll_cash,coll_card,danger} ||  
  Assess
```


MC Attention Response

- machine that delivers cash first
 - meets user expectation

MC Attention Response

- machine that delivers cash first
 - meets user expectation \implies MC: No

MC Attention Response

- machine that delivers cash first
 - meets user expectation \implies MC: No
 - doesn't meet user expectation

MC Attention Response

- machine that delivers cash first
 - meets user expectation \implies MC: No
 - doesn't meet user expectation \implies MC: No

MC Attention Response

- machine that delivers cash first
 - meets user expectation \implies MC: No
 - doesn't meet user expectation \implies MC: No
- machine that delivers card first
 - meets user expectation

MC Attention Response

- machine that delivers cash first
 - meets user expectation \implies MC: No
 - doesn't meet user expectation \implies MC: No
- machine that delivers card first
 - meets user expectation \implies MC: Yes

MC Attention Response

- machine that delivers cash first
 - meets user expectation \implies MC: No
 - doesn't meet user expectation \implies MC: No
- machine that delivers card first
 - meets user expectation \implies MC: Yes
 - doesn't meet user expectation

MC Attention Response

- machine that delivers cash first
 - meets user expectation \implies MC: No
 - doesn't meet user expectation \implies MC: No
- machine that delivers card first
 - meets user expectation \implies MC: Yes
 - doesn't meet user expectation \implies MC: No

MC Attention Response

- machine that delivers cash first
 - meets user expectation \implies MC: No
 - doesn't meet user expectation \implies MC: No
- machine that delivers card first
 - meets user expectation \implies MC: Yes
 - doesn't meet user expectation \implies MC: No

Why?

MC Attention Response

- machine that delivers cash first
 - meets user expectation \implies MC: No
 - doesn't meet user expectation \implies MC: No
- machine that delivers card first
 - meets user expectation \implies MC: Yes
 - doesn't meet user expectation \implies MC: No

Why?

Because by receiving the card instead of the expected cash, the user believes the card has been rejected and is in danger of being confiscated if used again

History of Formal HCI

Safety Motivation

History of Formal HCI

Safety Motivation

- **1980s: Human Reliability Assessment**
techniques [Svenson 1989, Kirwan 1990]

History of Formal HCI

Safety Motivation

- **1980s: Human Reliability Assessment** techniques [Svenson 1989, Kirwan 1990]
- **1990s: Formal Methods** techniques for the analysis of
 - **expected effective operator behaviour** [Liskov and Wing 1994, Leveson 1990]
 - **errors effectively performed by the operator** [Johnson 1997]

History of Formal HCI

Safety Motivation

- **1980s: Human Reliability Assessment** techniques [Svenson 1989, Kirwan 1990]
- **1990s: Formal Methods** techniques for the analysis of
 - **expected effective operator behaviour** [Liskov and Wing 1994, Leveson 1990]
 - **errors effectively performed by the operator** [Johnson 1997]

But human behaviour is **unpredictable**

History of Formal HCI (cont.)

Unpredictable Behaviour

History of Formal HCI (cont.)

Unpredictable Behaviour

- **end 1990s: Cognitively Plausible Behaviour**
[Butler et al. 1998, Butterworth et al. 2000, Rushby 2002, Curzon and Blandford 2004]

History of Formal HCI (cont.)

Unpredictable Behaviour

- **end 1990s: Cognitively Plausible Behaviour**
[Butler et al. 1998, Butterworth et al. 2000, Rushby 2002, Curzon and Blandford 2004]

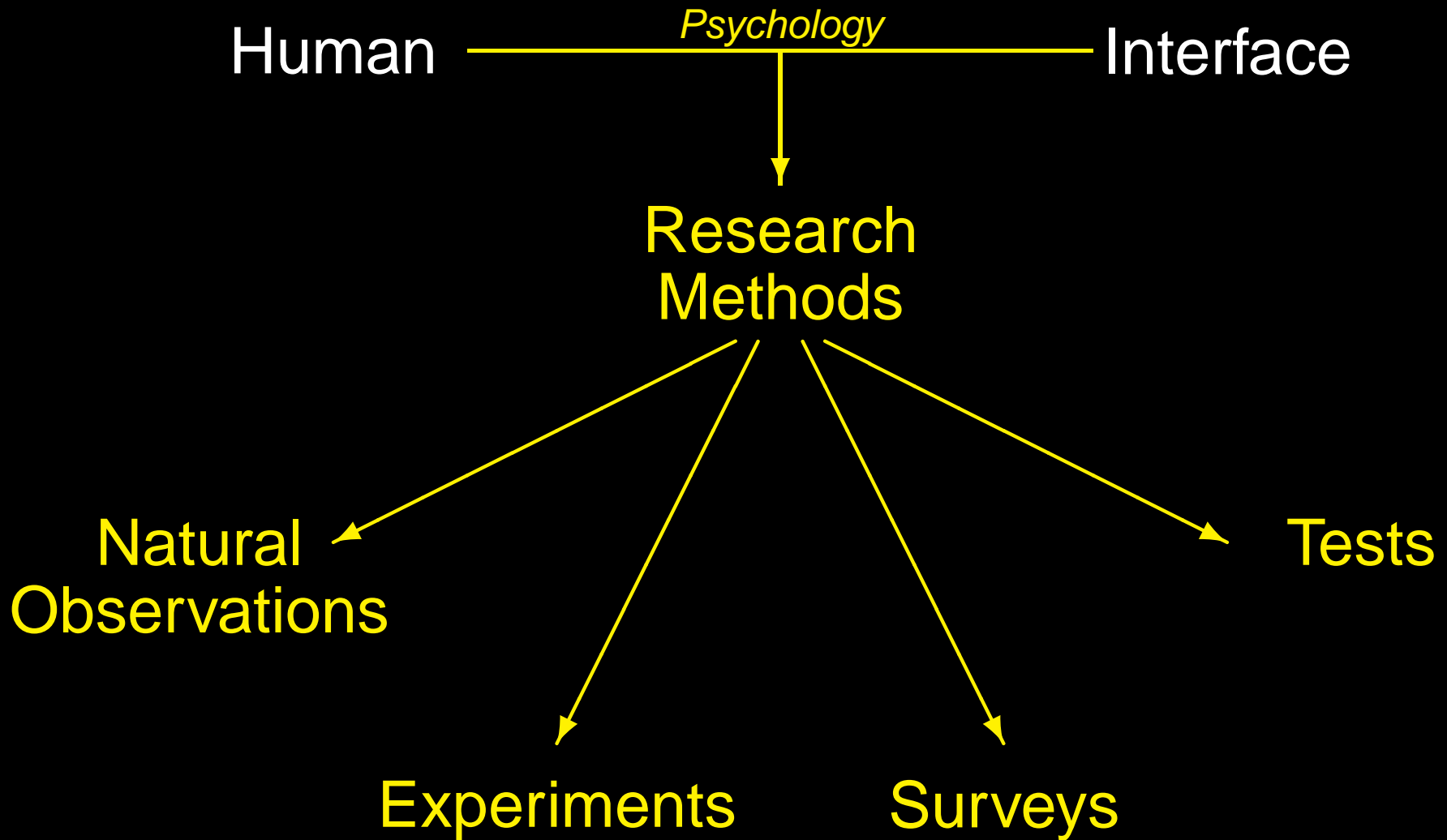
Security Motivation

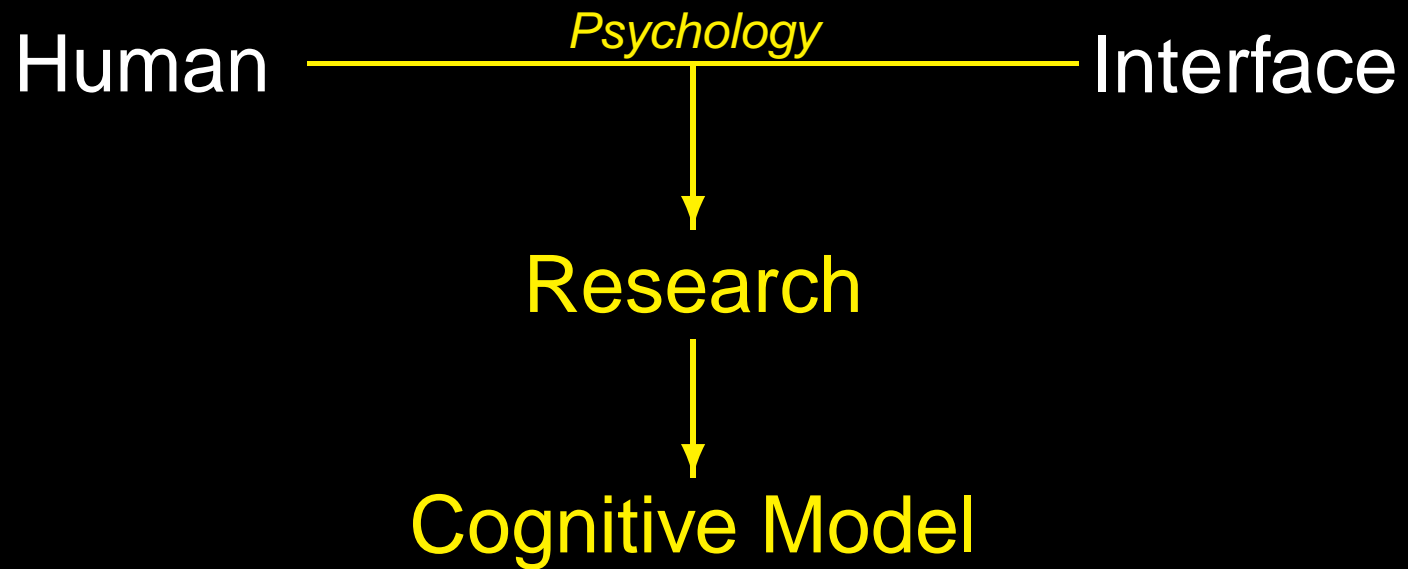
- **2000s: Usability affects Security** [Zurko 2005, Cerone and Curzon 2007]

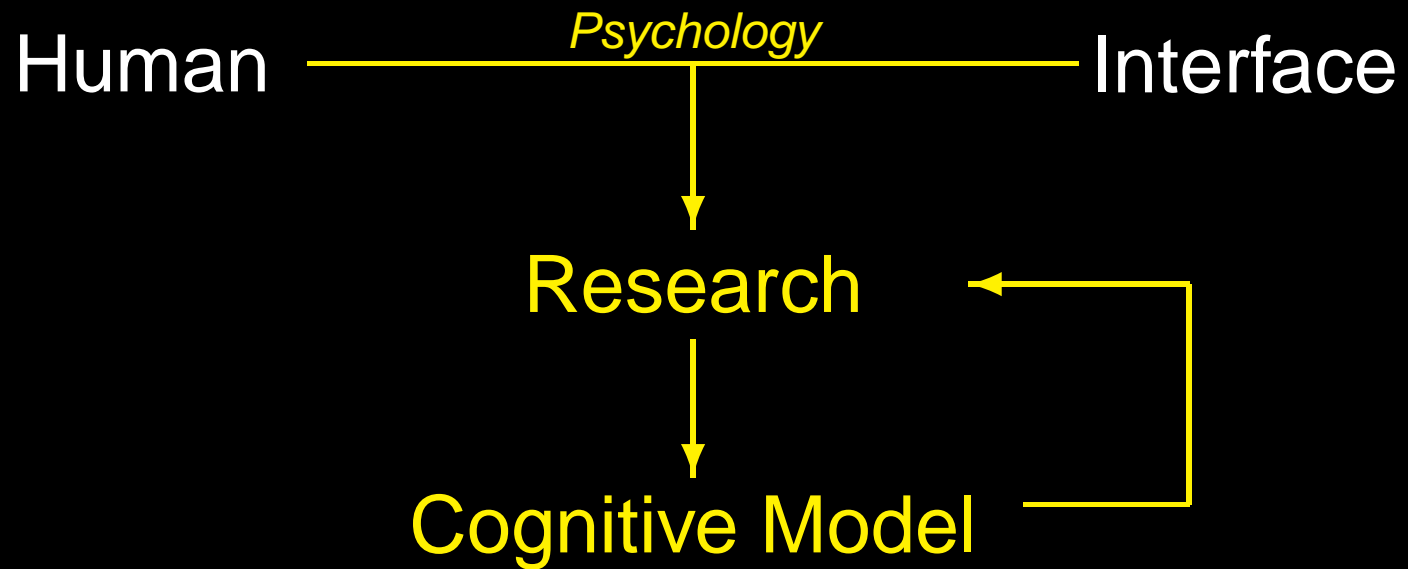
Task Failure and ATC Example

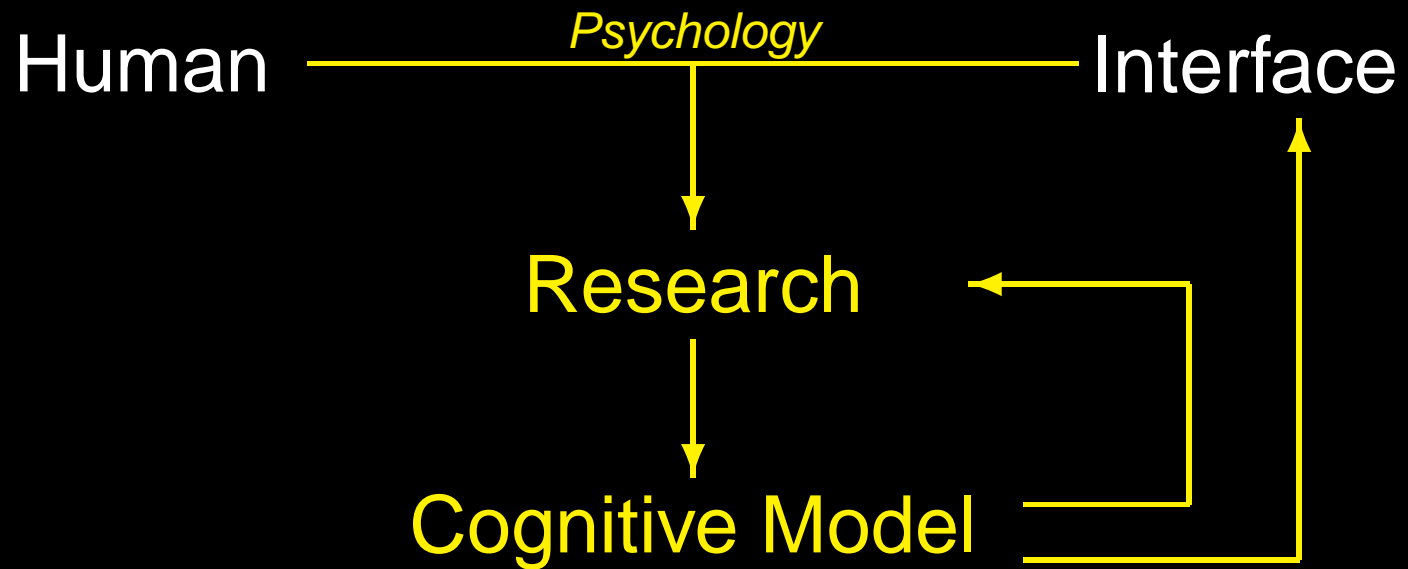
Human

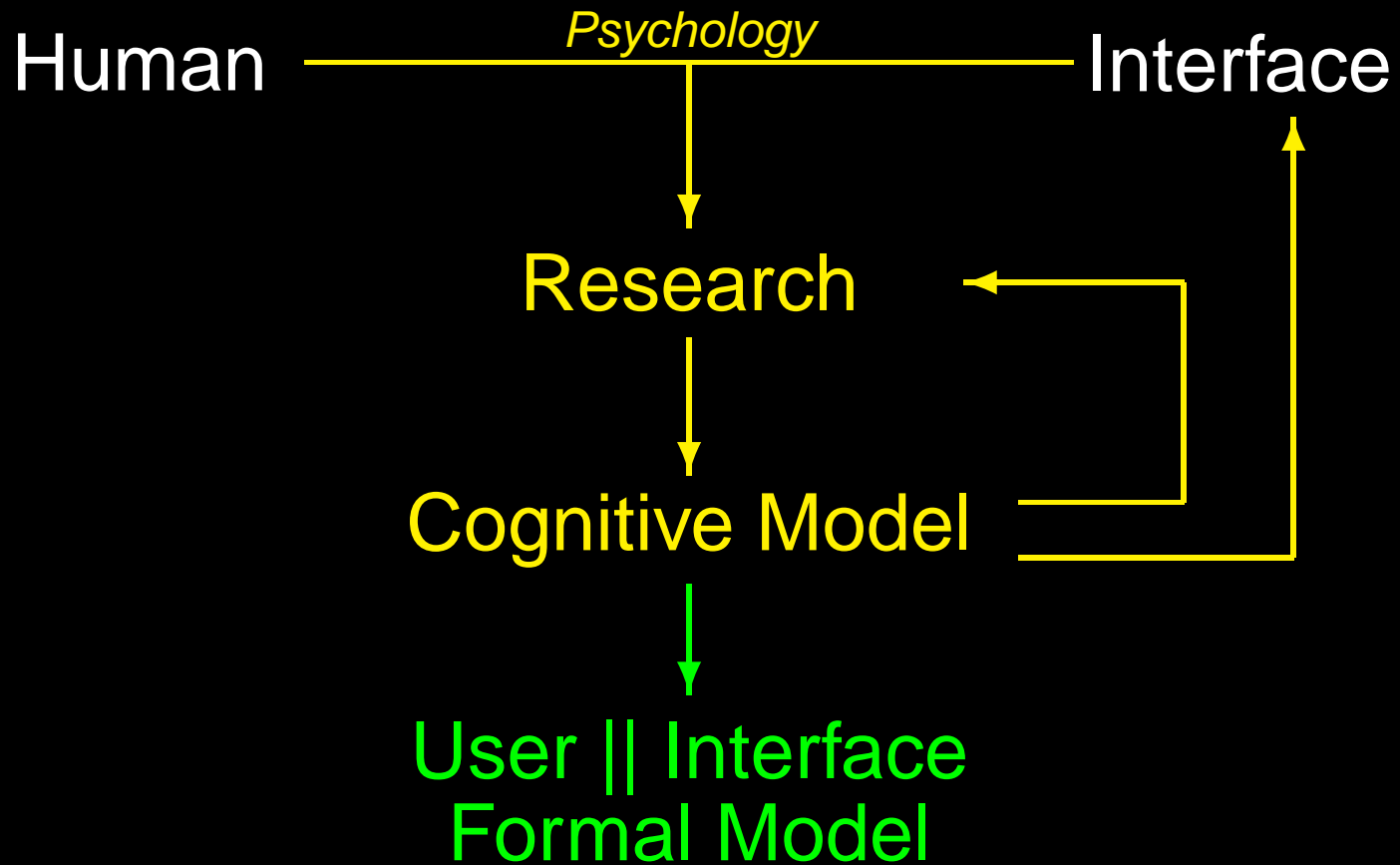
Interface

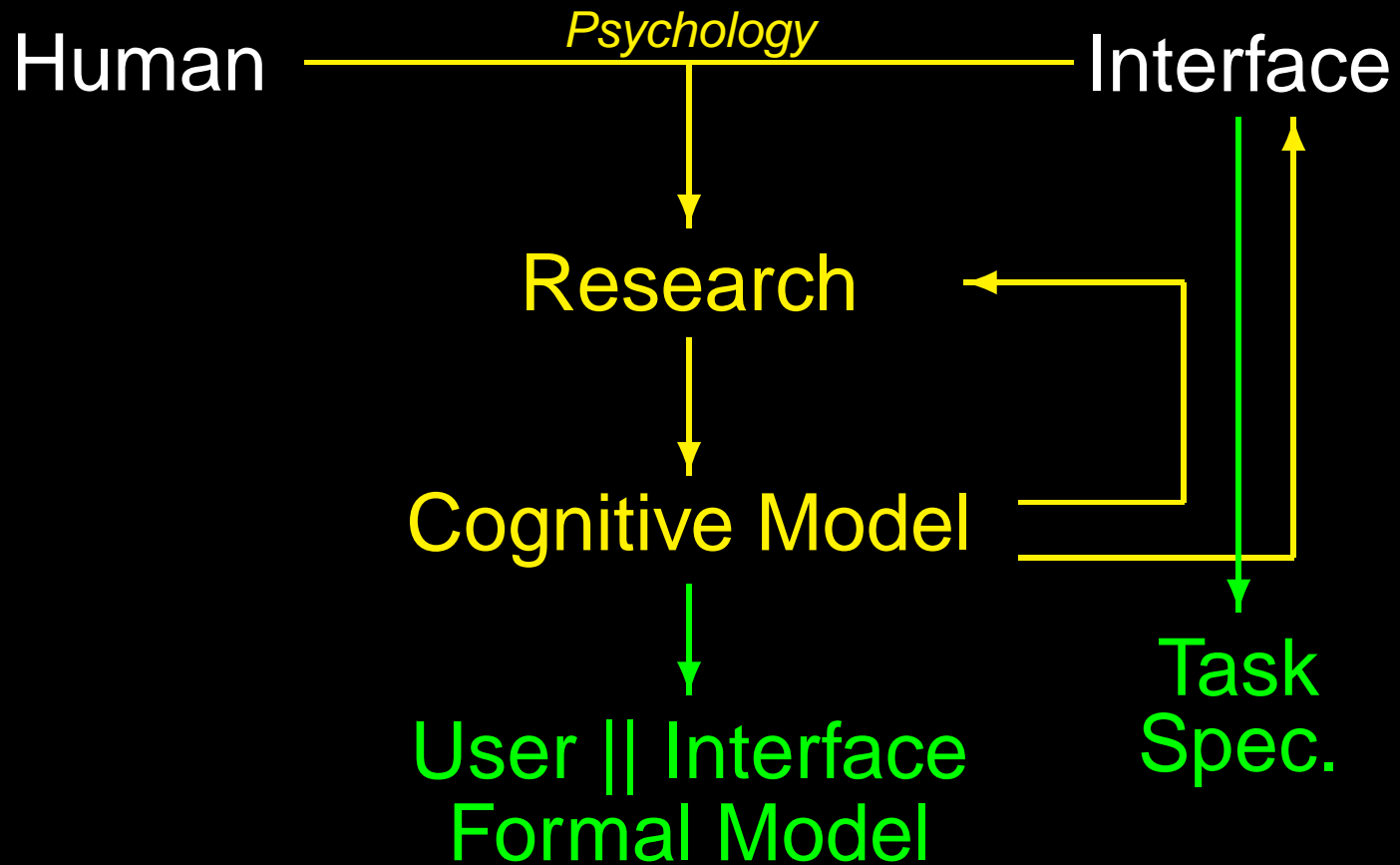


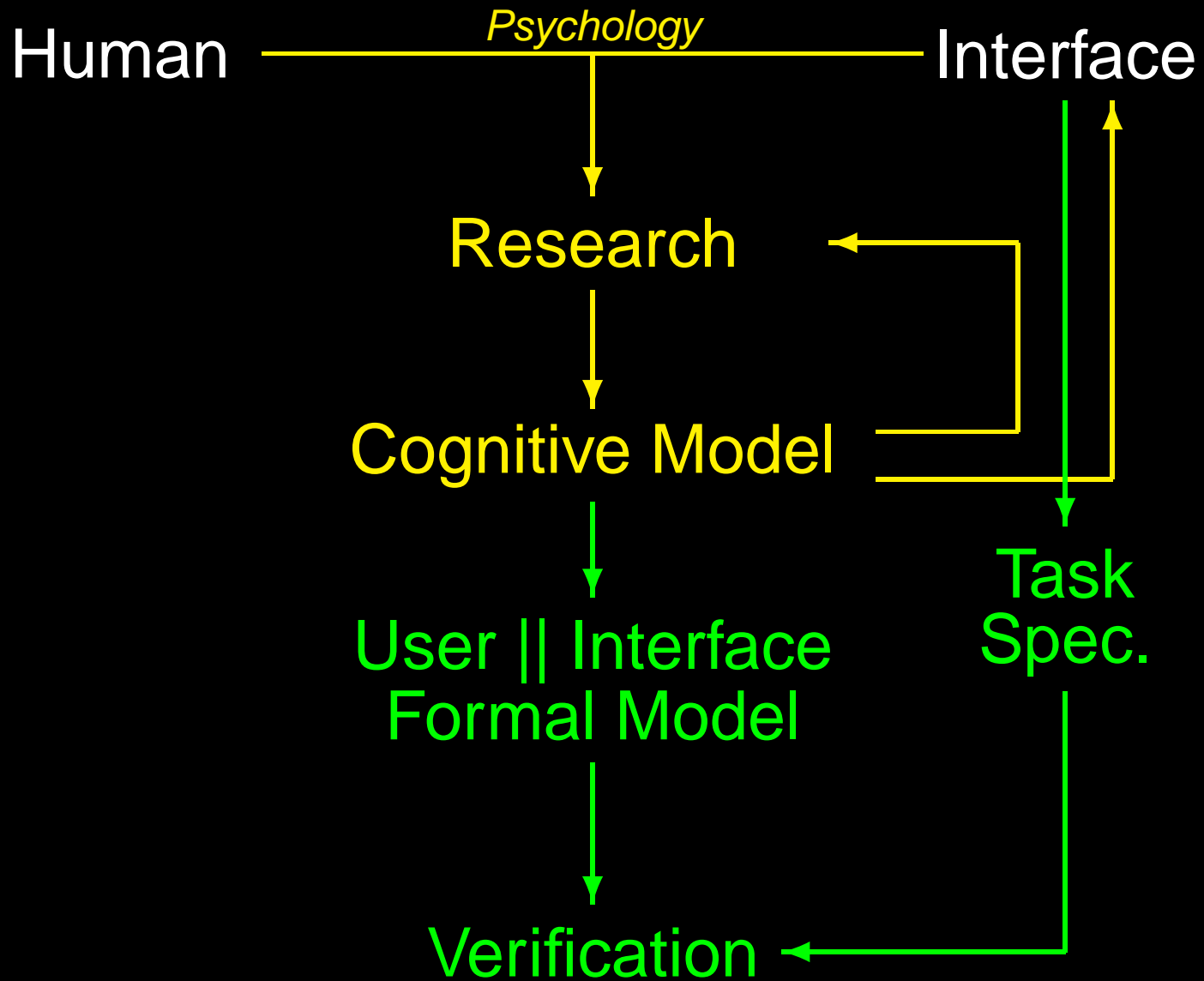


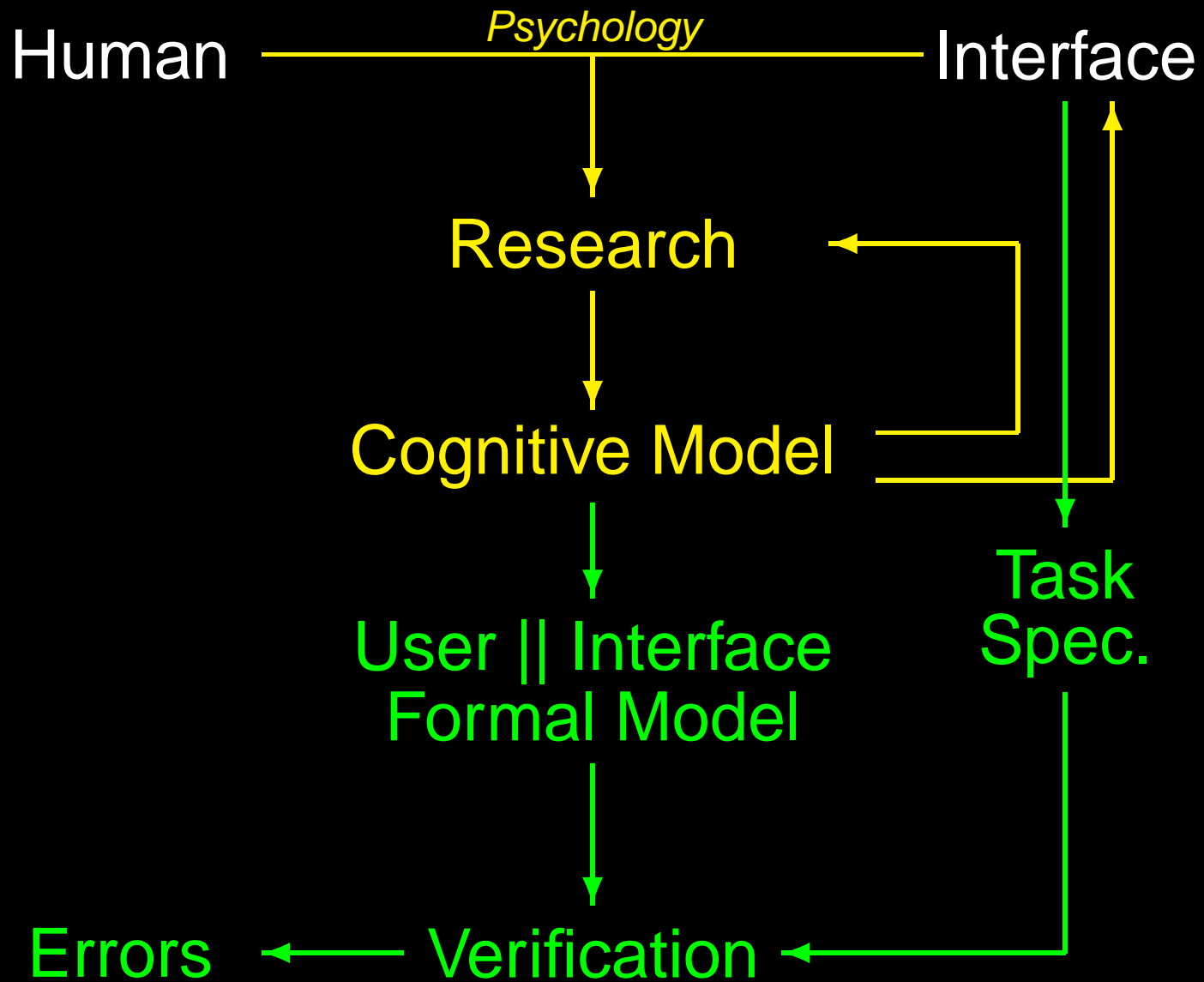


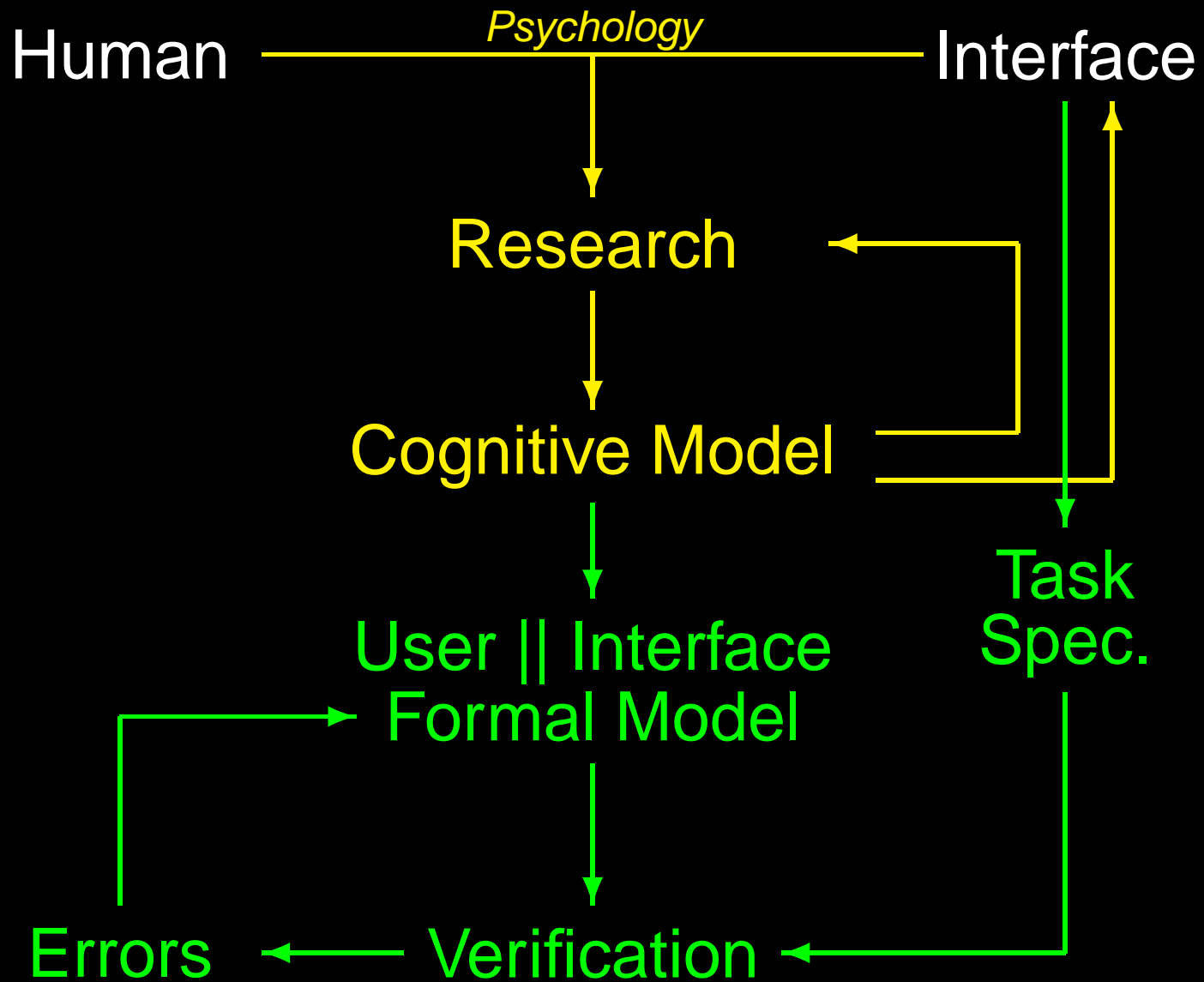


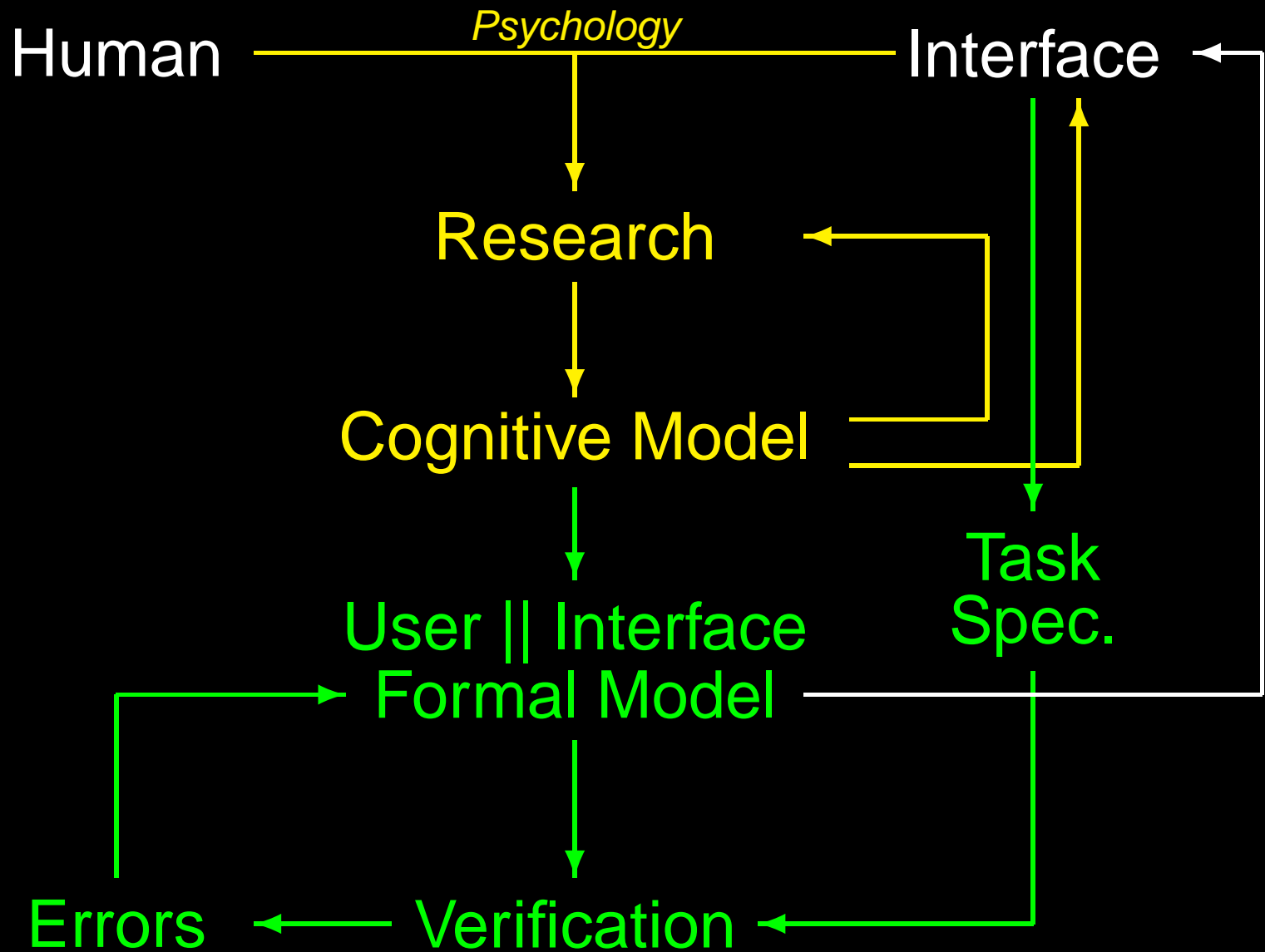












ATM Properties in LTL

Functional Correctness:

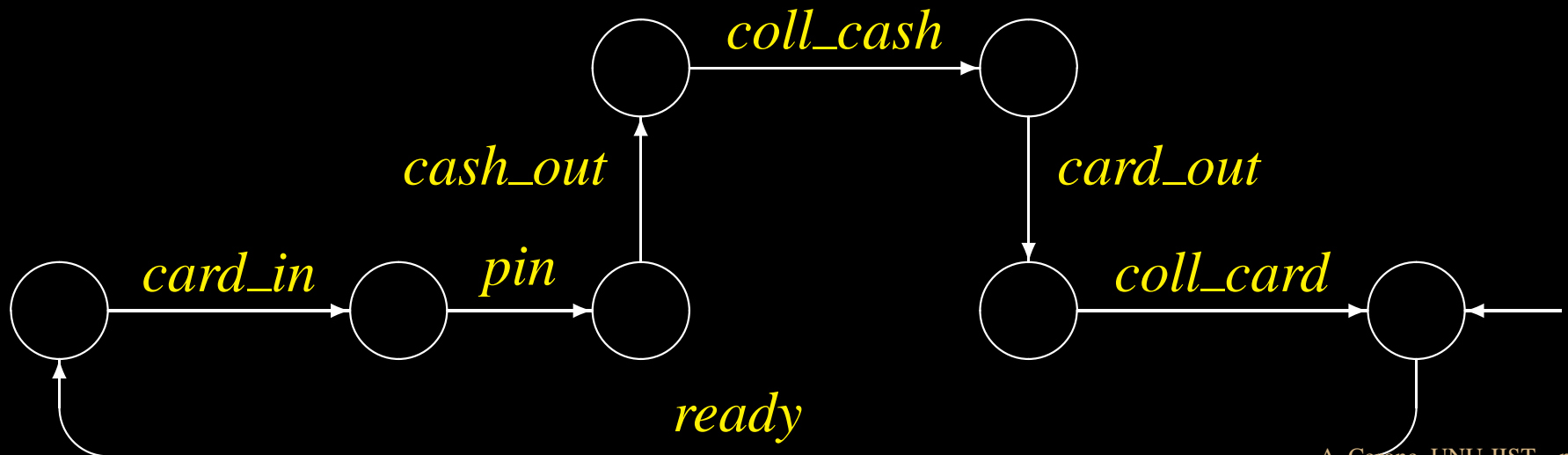
The ATM machine will eventually deliver cash

$$\Box(\text{ready} \rightarrow \Diamond \text{cash_out})$$

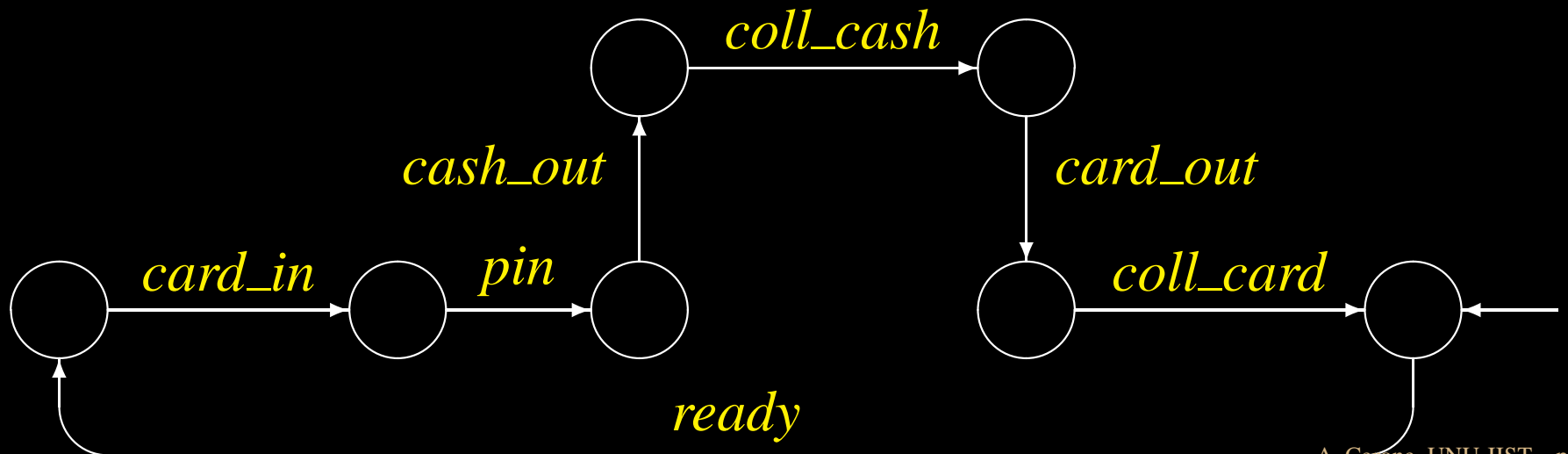
Safety:

The ATM machine will eventually return the card

$$\Box(\text{ready} \rightarrow \Diamond \text{card_out})$$

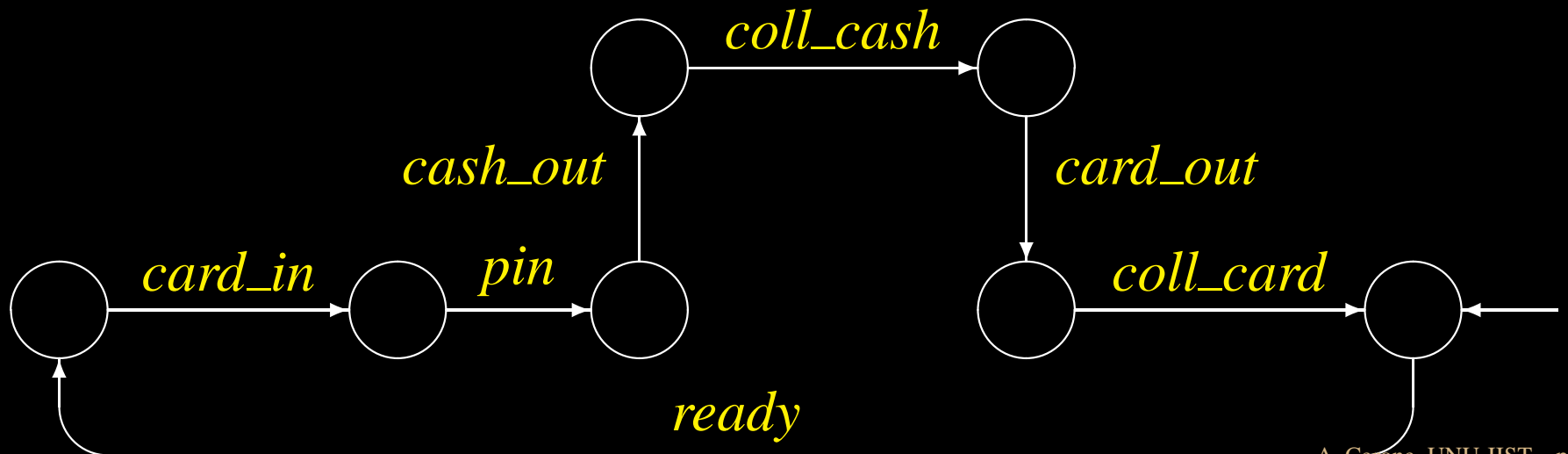


Example: ATM Machine



Example: ATM Machine

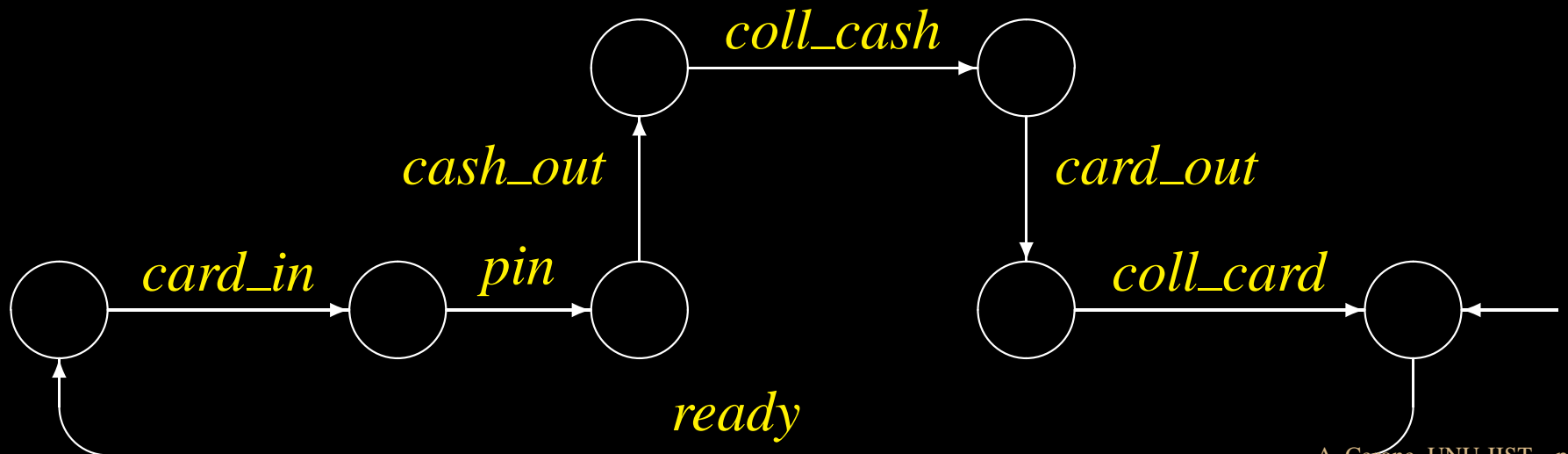
Goal: $\Box(\text{ready} \rightarrow \Diamond \text{coll_cash})$



Example: ATM Machine

Goal: $\Box(\text{ready} \rightarrow \Diamond \text{coll_cash})$

Safety: $\Box(\text{ready} \rightarrow \Diamond \text{coll_card})$

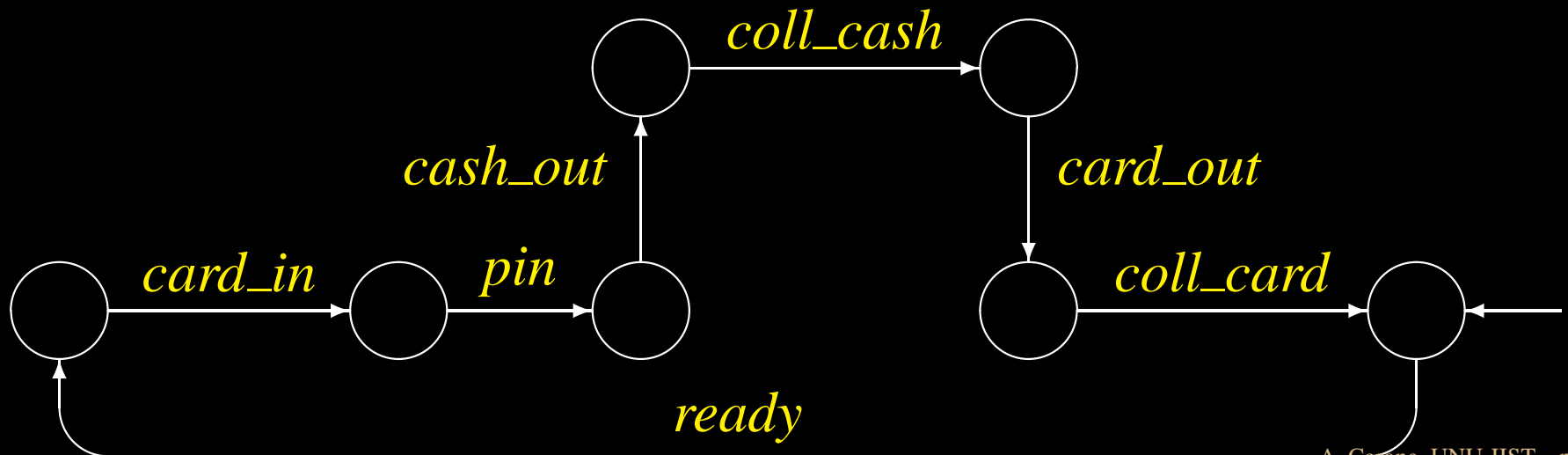


Example: ATM Machine

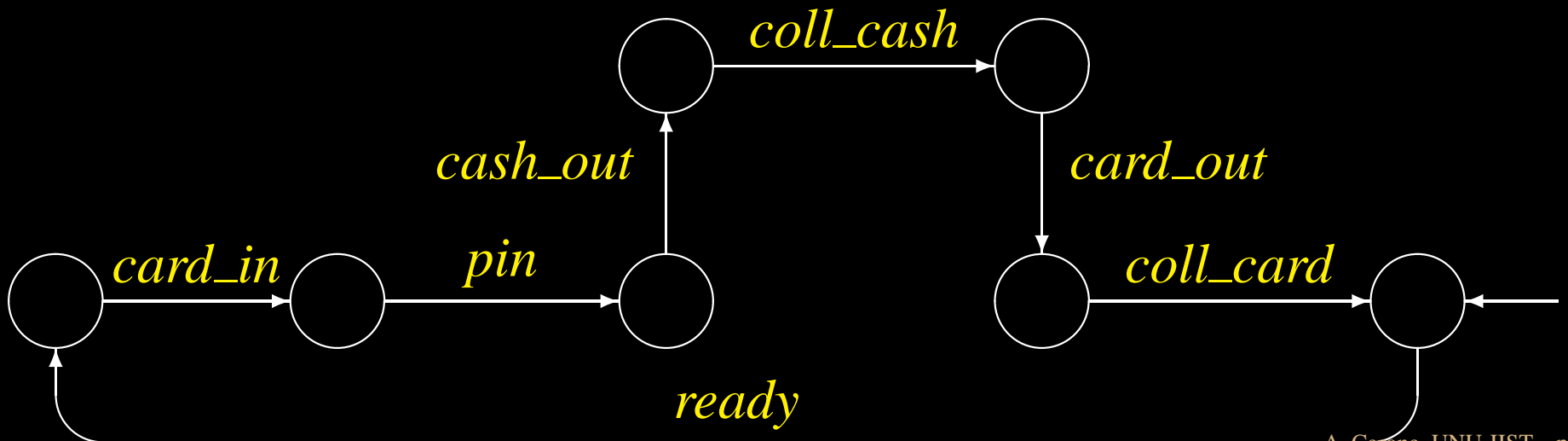
Goal: $\Box(\text{ready} \rightarrow \Diamond \text{coll_cash})$

Safety: $\Box(\text{ready} \rightarrow \Diamond \text{coll_card})$

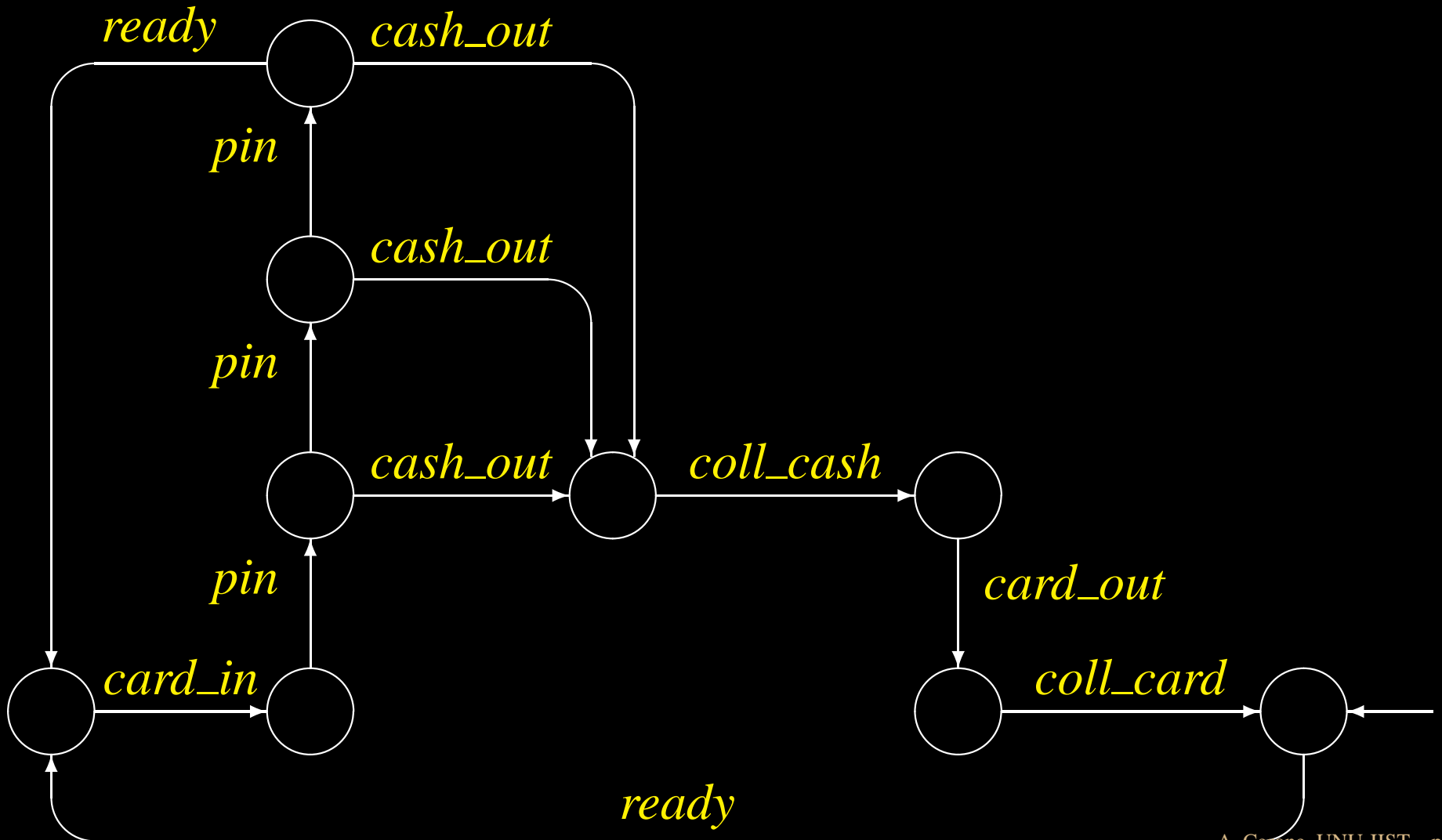
Task: $\Box(\text{ready} \rightarrow ((\Diamond \text{coll_cash}) \wedge (\Diamond \text{coll_card})))$



Refined ATM Machine



Refined ATM Machine



ATM: Task Failure

Goal: $\Box(\text{ready} \rightarrow \Diamond \text{coll_cash})$

Safety: $\Box(\text{ready} \rightarrow \Diamond \text{coll_card})$

Task: $\Box(\text{ready} \rightarrow ((\Diamond \text{coll_cash}) \wedge (\Diamond \text{coll_card})))$

ATM: Task Failure

Goal: $\Box(\text{ready} \rightarrow \Diamond \text{coll_cash})$

Safety: $\Box(\text{ready} \rightarrow \Diamond \text{coll_card})$

Task: $\Box(\text{ready} \rightarrow ((\Diamond \text{coll_cash}) \wedge (\Diamond \text{coll_card})))$

Task Failure:

$\neg \Box((\Diamond \text{coll_cash}) \wedge (\Diamond \text{coll_card}))$

ATM: Task Failure

Goal: $\Box(\text{ready} \rightarrow \Diamond \text{coll_cash})$

Safety: $\Box(\text{ready} \rightarrow \Diamond \text{coll_card})$

Task: $\Box(\text{ready} \rightarrow ((\Diamond \text{coll_cash}) \wedge (\Diamond \text{coll_card})))$

Task Failure:

$$\Diamond((\Box \neg \text{coll_cash}) \vee (\Box \neg \text{coll_card}))$$

ATM: Task Failure

Goal: $\Box(\text{ready} \rightarrow \Diamond \text{coll_cash})$

Safety: $\Box(\text{ready} \rightarrow \Diamond \text{coll_card})$

Task: $\Box(\text{ready} \rightarrow ((\Diamond \text{coll_cash}) \wedge (\Diamond \text{coll_card})))$

Task Failure:

$(\Box \neg \text{coll_cash}) \vee (\Box \neg \text{coll_card})$

Task Failure Decomposition

Top-level Task Failure:

$$(\Box \neg coll_cash) \vee (\Box \neg coll_card))$$

Task Failure Decomposition

Top-level Task Failure:

$$(\Box \neg coll_cash) \vee (\Box \neg coll_card))$$

1. input wrong pin three times in a row
 \implies card confiscated and cash not collected

Task Failure Decomposition

Top-level Task Failure:

$$(\Box \neg coll_cash) \vee (\Box \neg coll_card)$$

1. input wrong pin three times in a row
 \implies card confiscated and cash not collected
 $(\Box \neg coll_cash) \wedge (\Box \neg coll_card)$

Task Failure Decomposition

Top-level Task Failure:

$$(\Box \neg coll_cash) \vee (\Box \neg coll_card))$$

1. input wrong pin three times in a row
 \implies card confiscated and cash not collected
 $(\Box \neg coll_cash) \wedge (\Box \neg coll_card))$
2. collect cash but not card

Task Failure Decomposition

Top-level Task Failure:

$$(\Box \neg coll_cash) \vee (\Box \neg coll_card))$$

1. input wrong pin three times in a row
 \implies card confiscated and cash not collected

$$(\Box \neg coll_cash) \wedge (\Box \neg coll_card))$$

2. collect cash but not card

$$(\Diamond coll_cash) \wedge (\Box \neg coll_card))$$

Task Failure Decomposition

Top-level Task Failure:

$$(\Box \neg coll_cash) \vee (\Box \neg coll_card))$$

1. input wrong pin three times in a row
 \implies card confiscated and cash not collected

$$(\Box \neg coll_cash) \wedge (\Box \neg coll_card))$$

2. collect cash but not card
 $(\Diamond coll_cash) \wedge (\Box \neg coll_card))$

3. collect card but not cash

Task Failure Decomposition

Top-level Task Failure:

$$(\Box \neg coll_cash) \vee (\Box \neg coll_card))$$

1. input wrong pin three times in a row
 \implies card confiscated and cash not collected

$$(\Box \neg coll_cash) \wedge (\Box \neg coll_card))$$

2. collect cash but not card
 $(\Diamond coll_cash) \wedge (\Box \neg coll_card))$

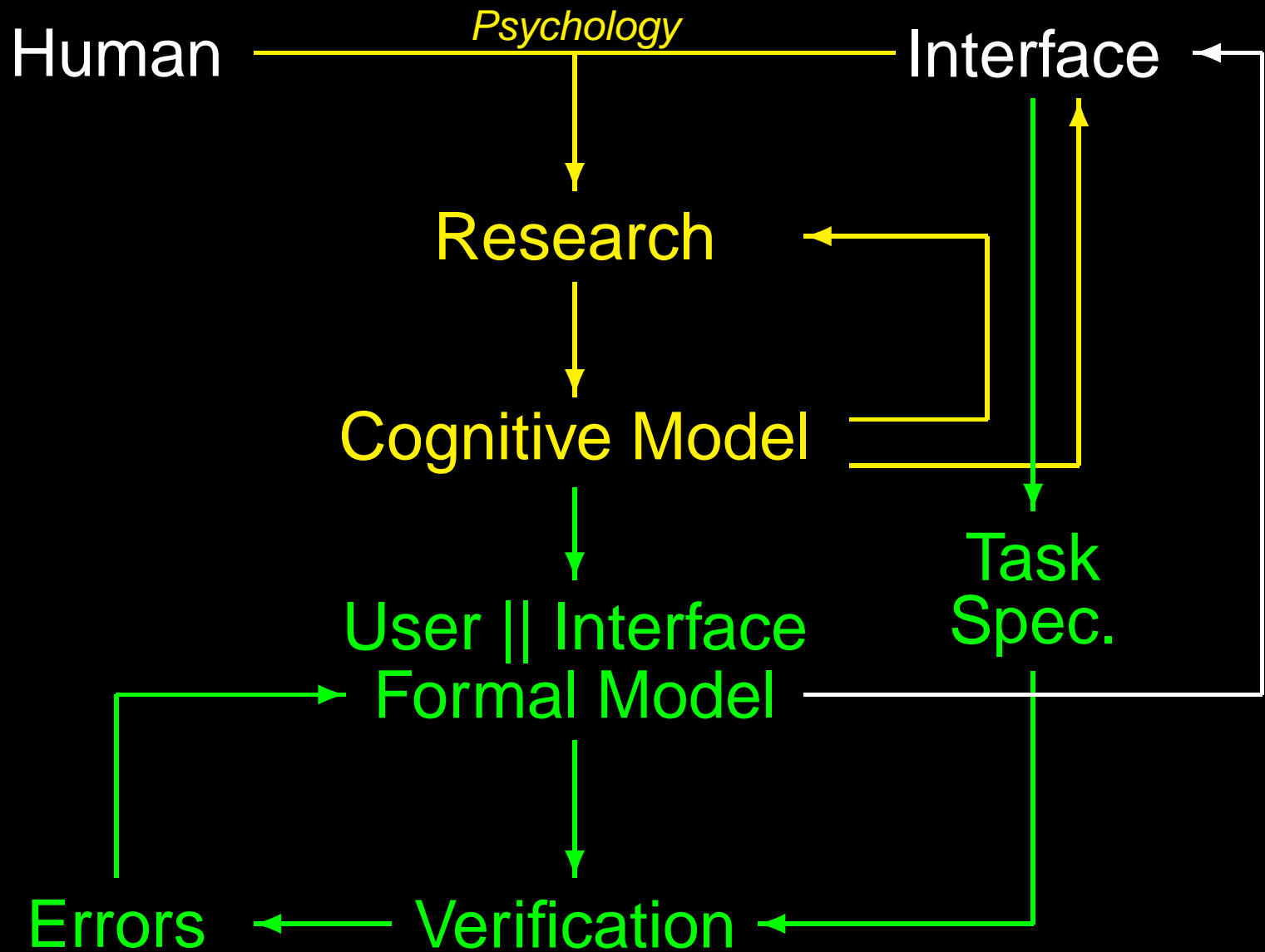
3. collect card but not cash
 $(\Diamond coll_card) \wedge (\Box \neg coll_cash))$

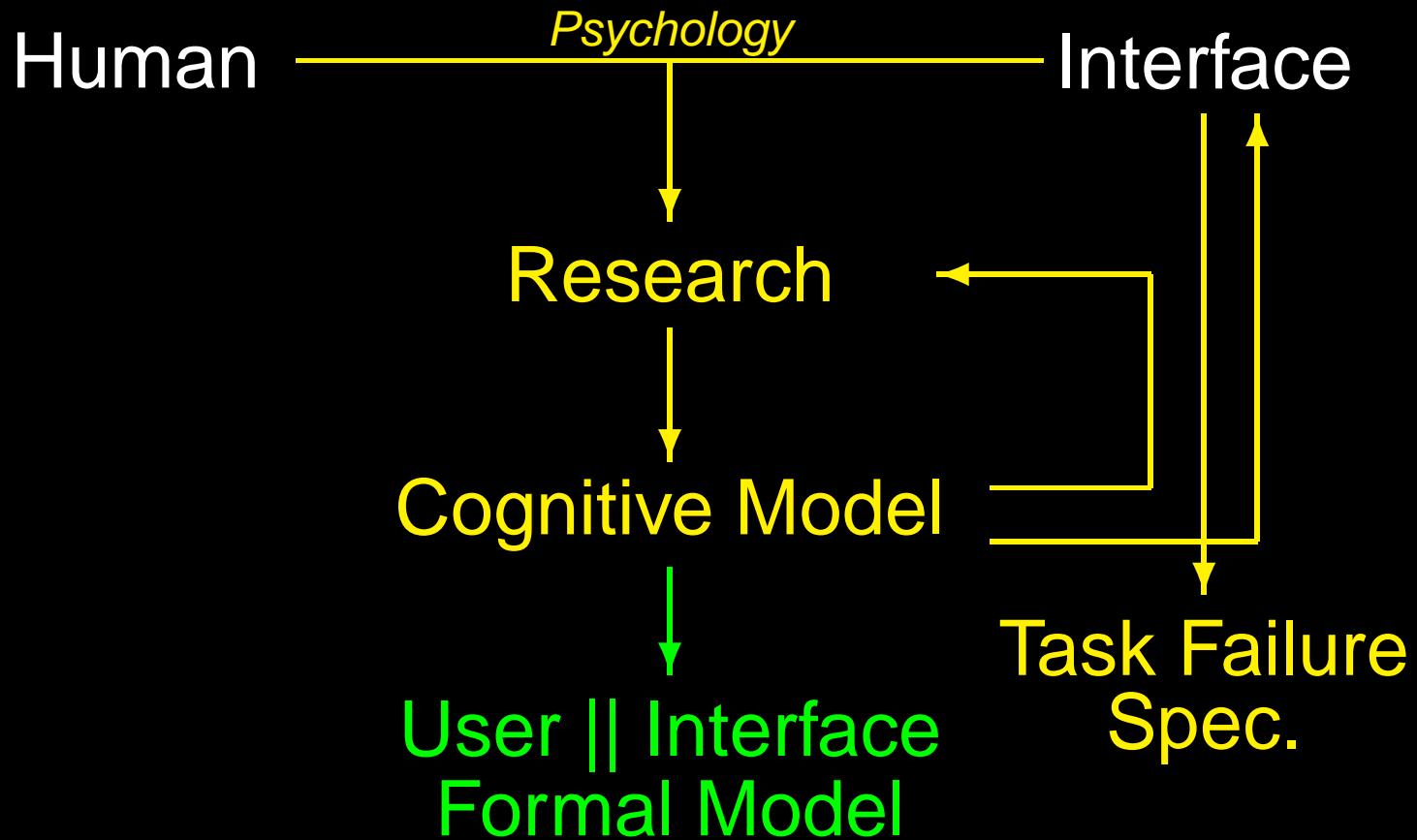
TF Psyc. Interpretation

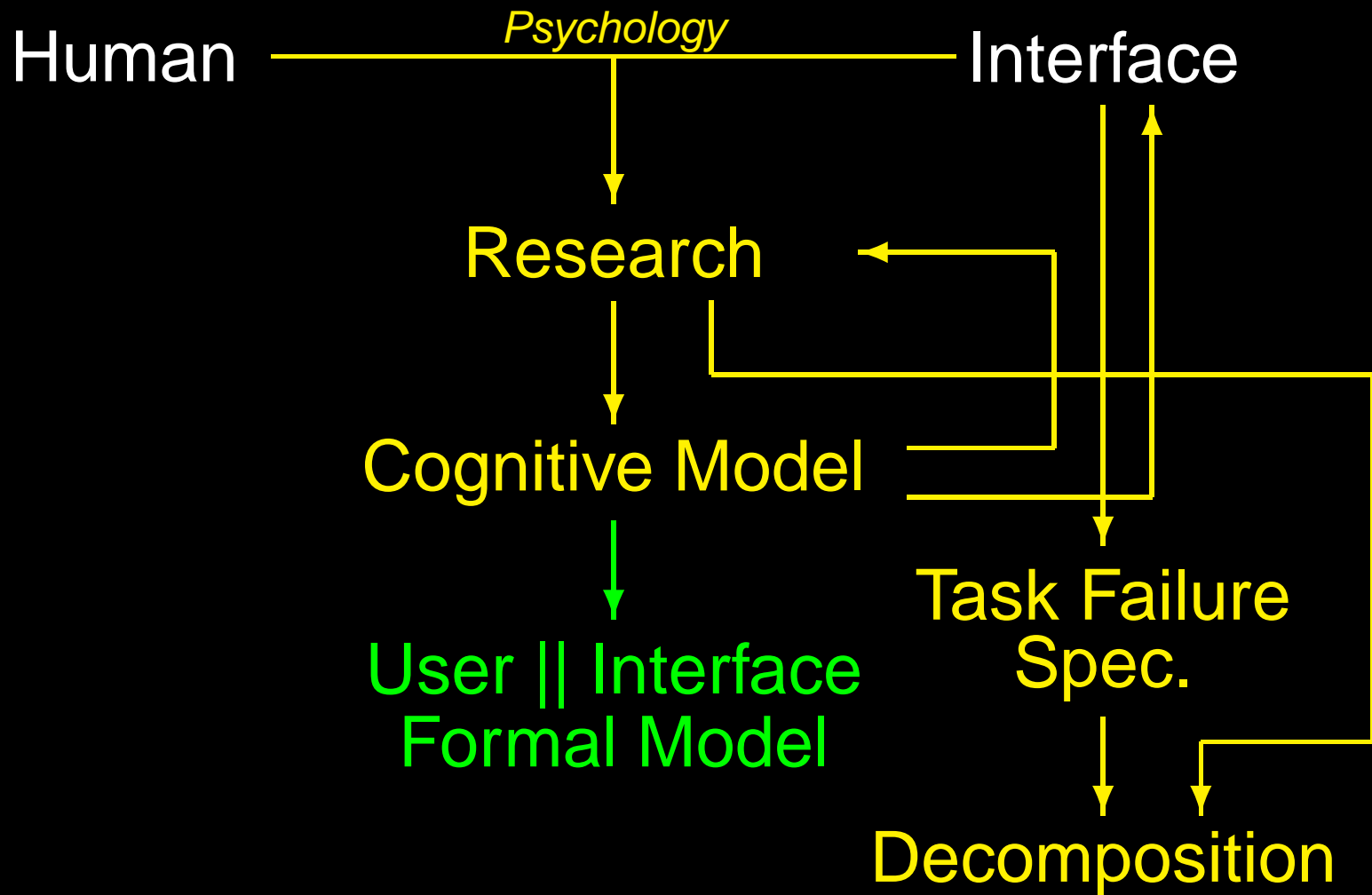
Top-level Task Failure:

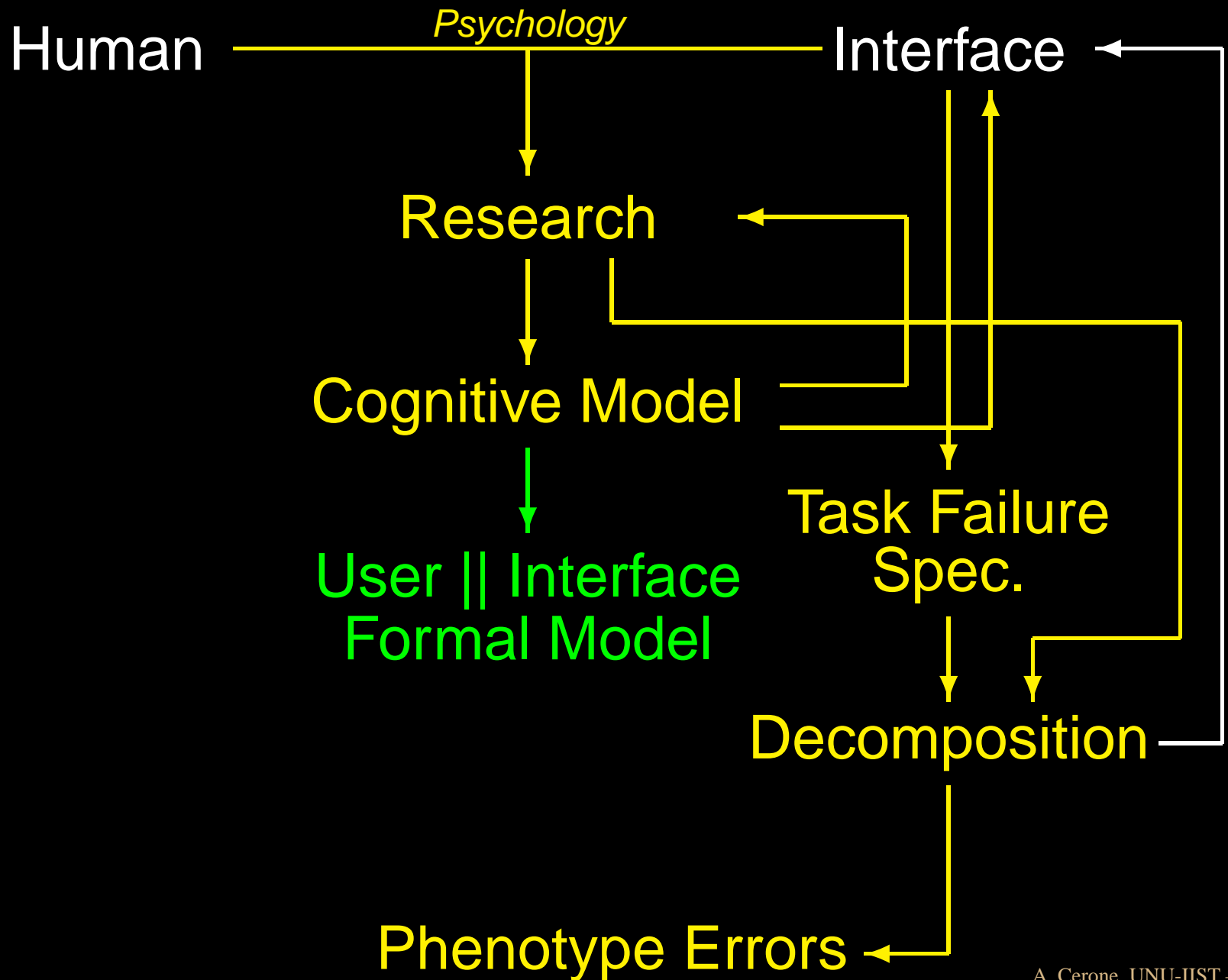
either card or cash is not collected

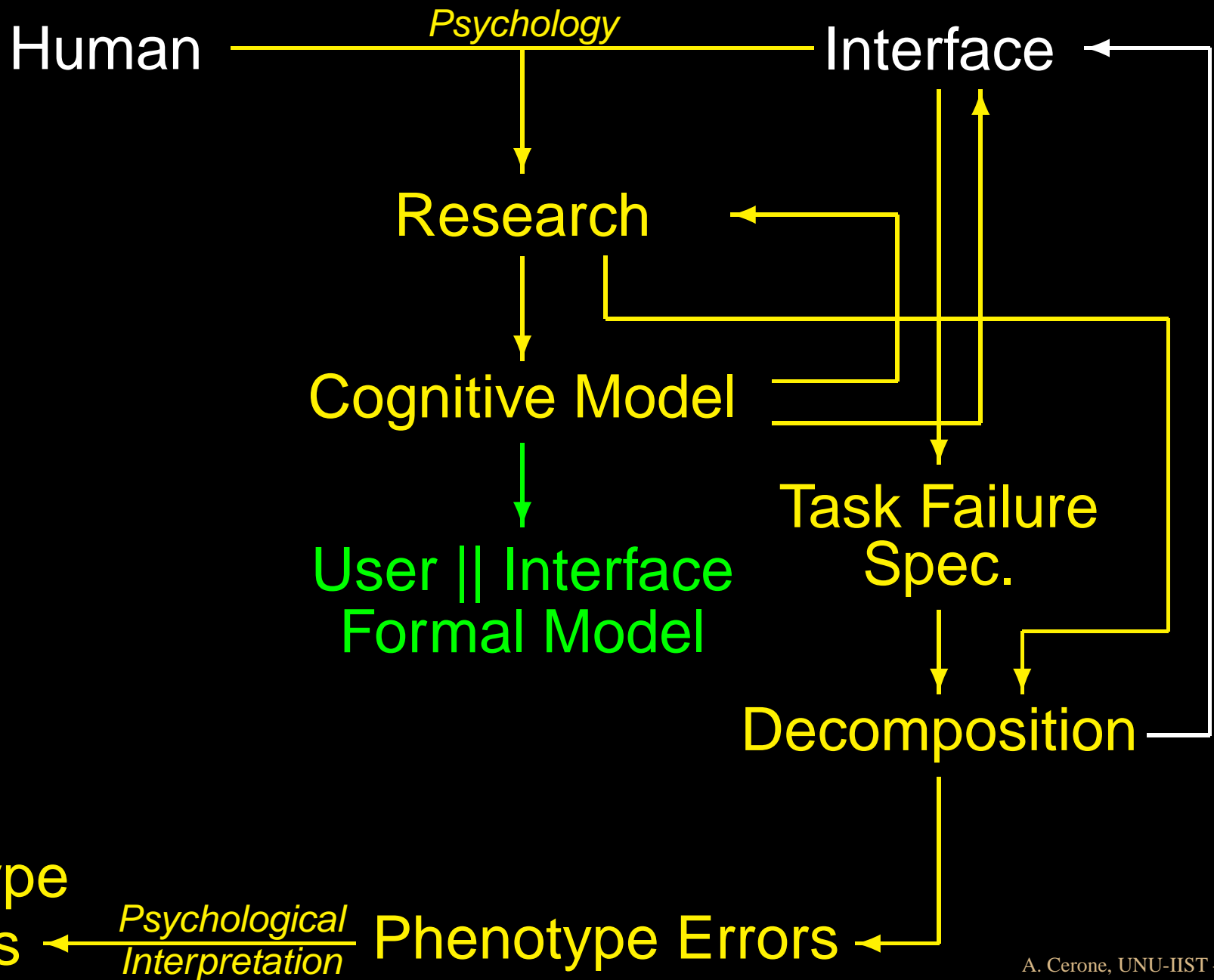
1. input wrong pin three times in a row
 - ⇒ card confiscated and cash not collected
 - ⇐ pin forgotten
2. collect cash but not card
 - ⇐ forget to collect card due to postcompletion error
3. collect card but not cash
 - ⇐ forget to collect cash

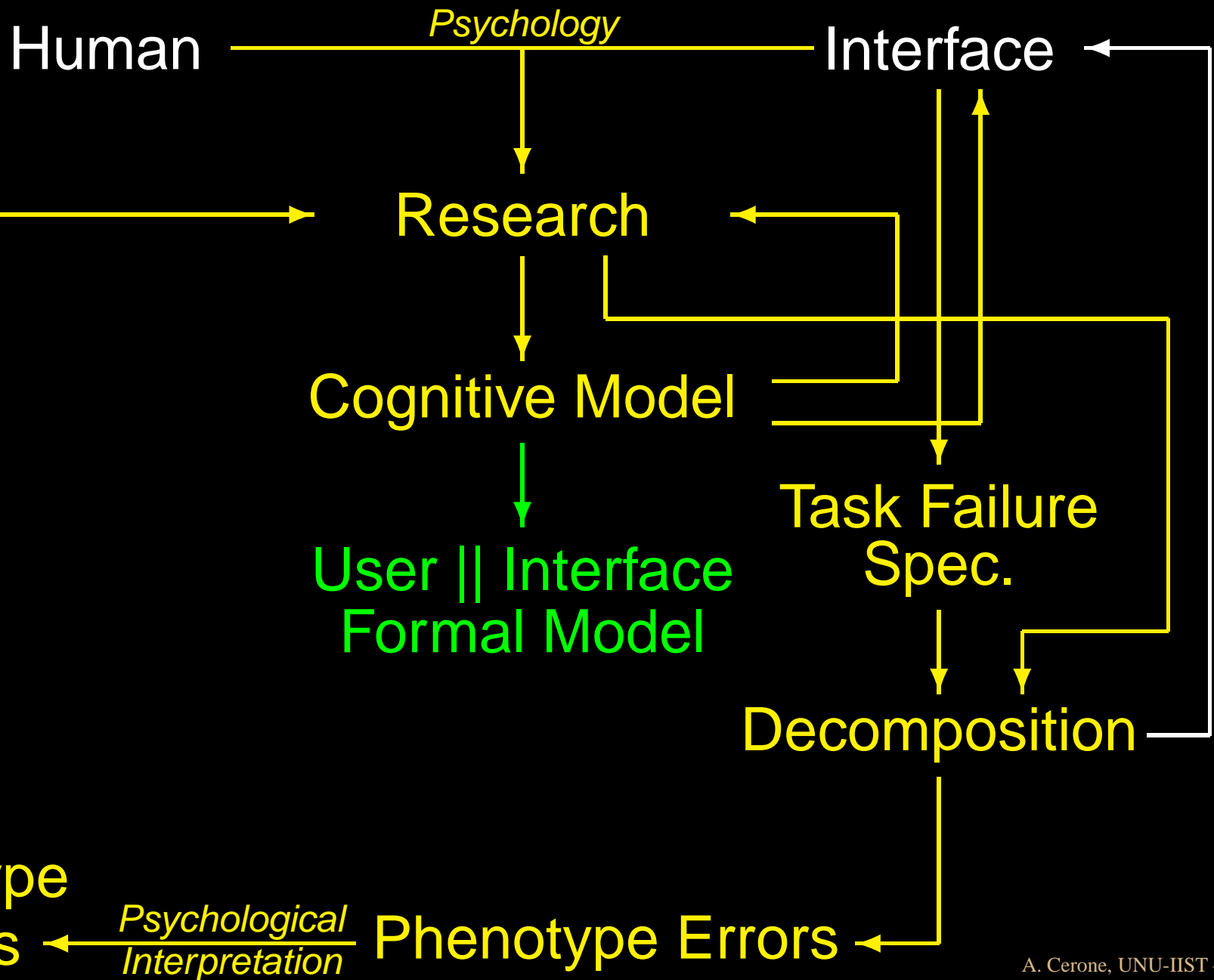


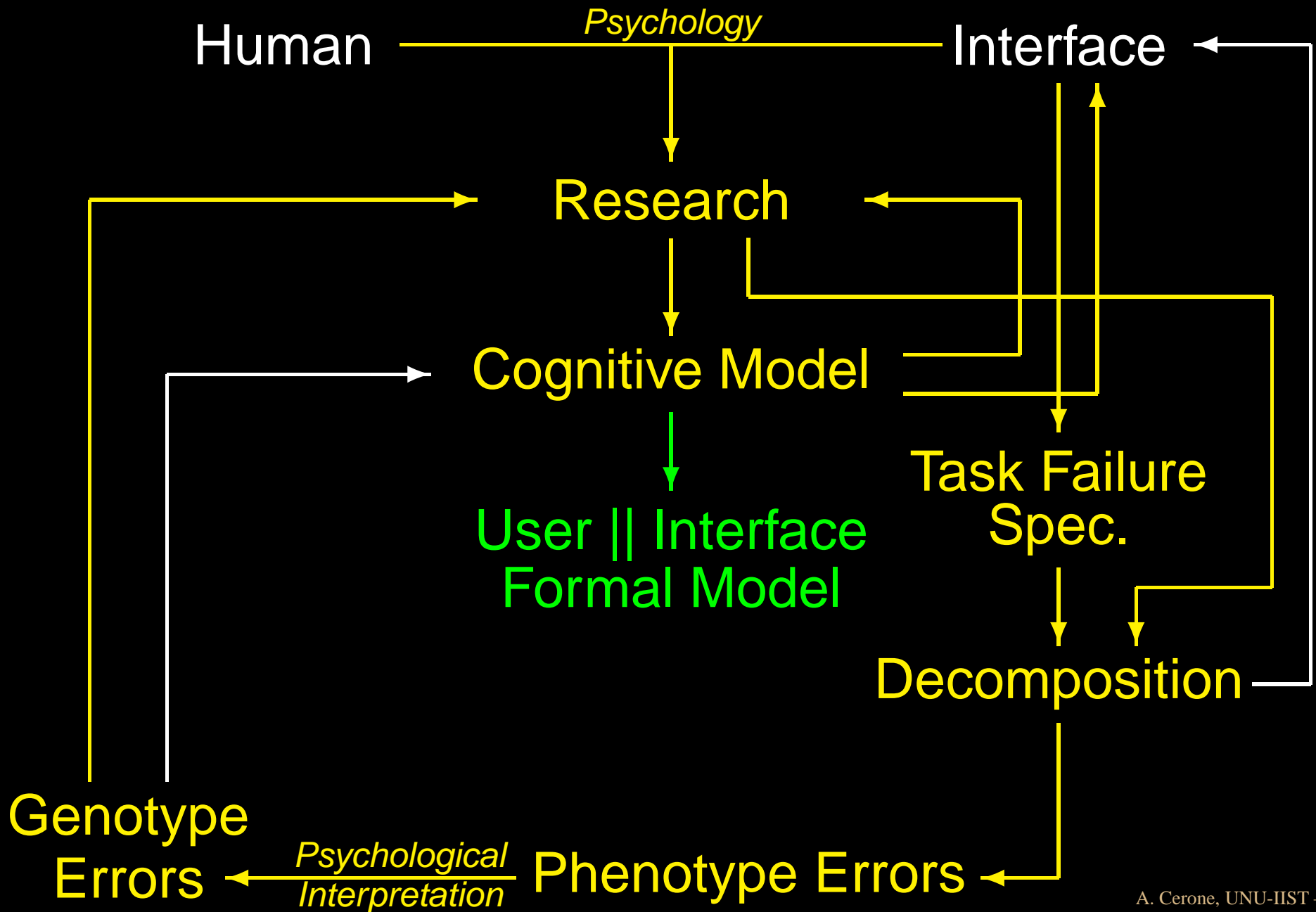


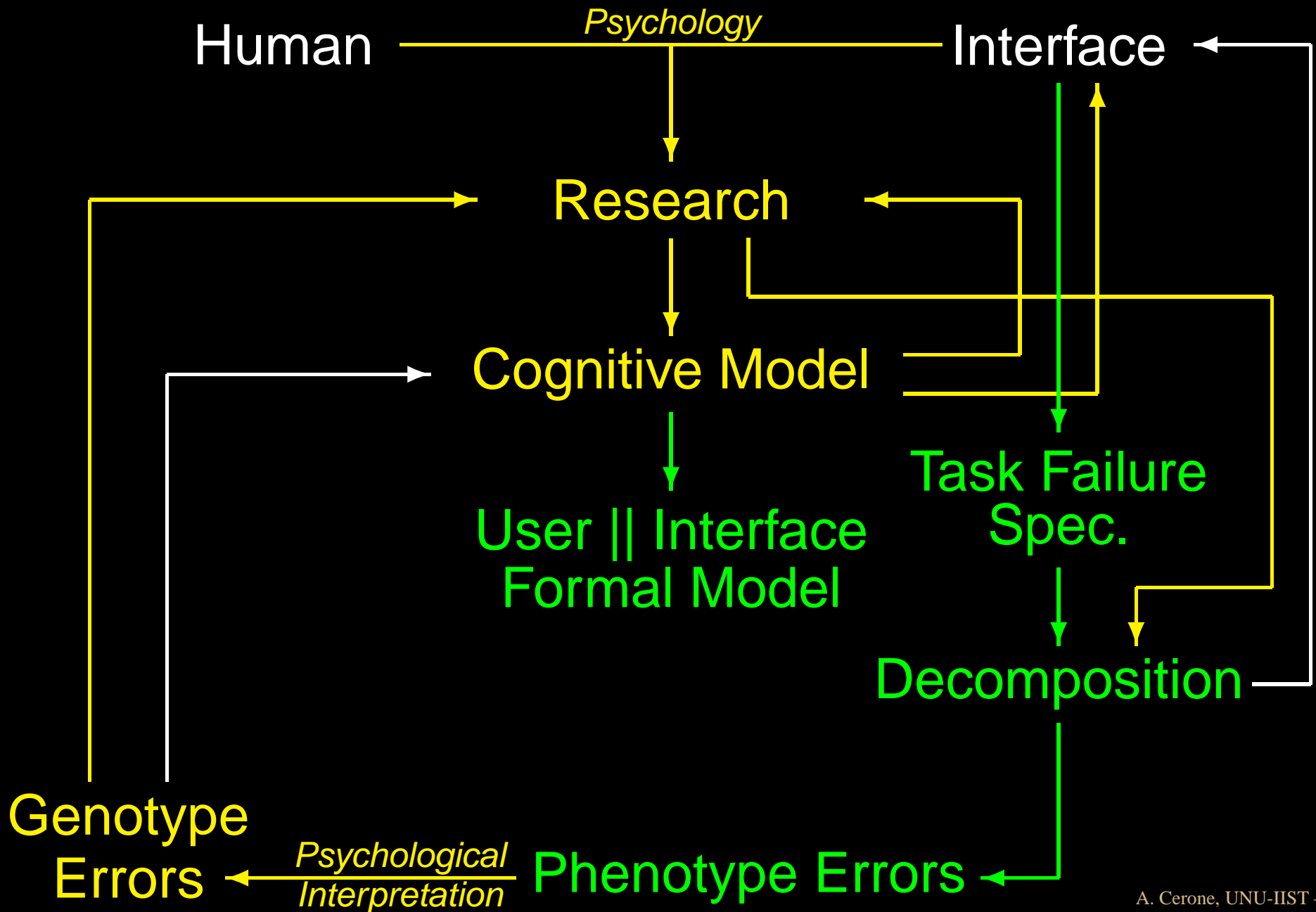


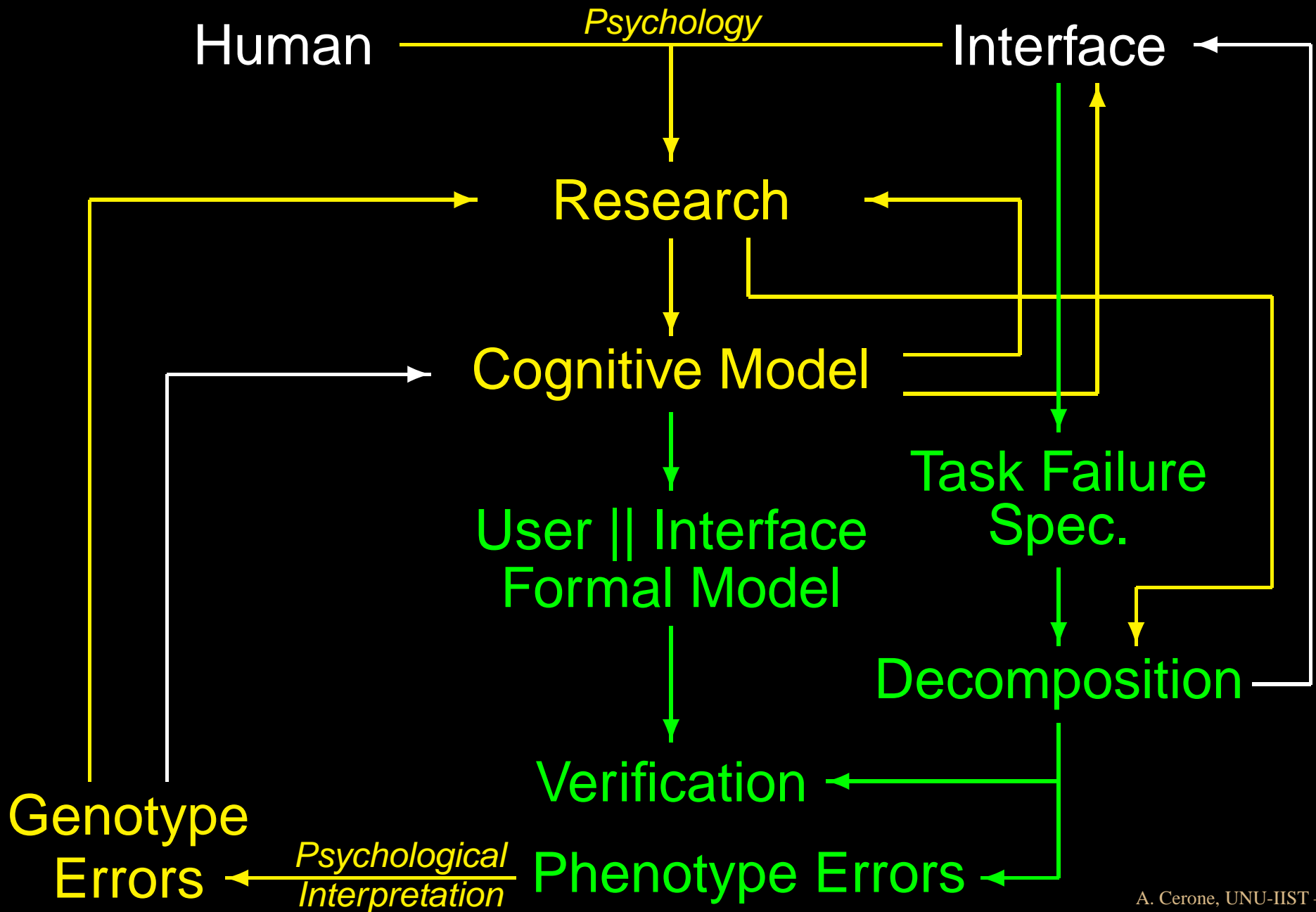


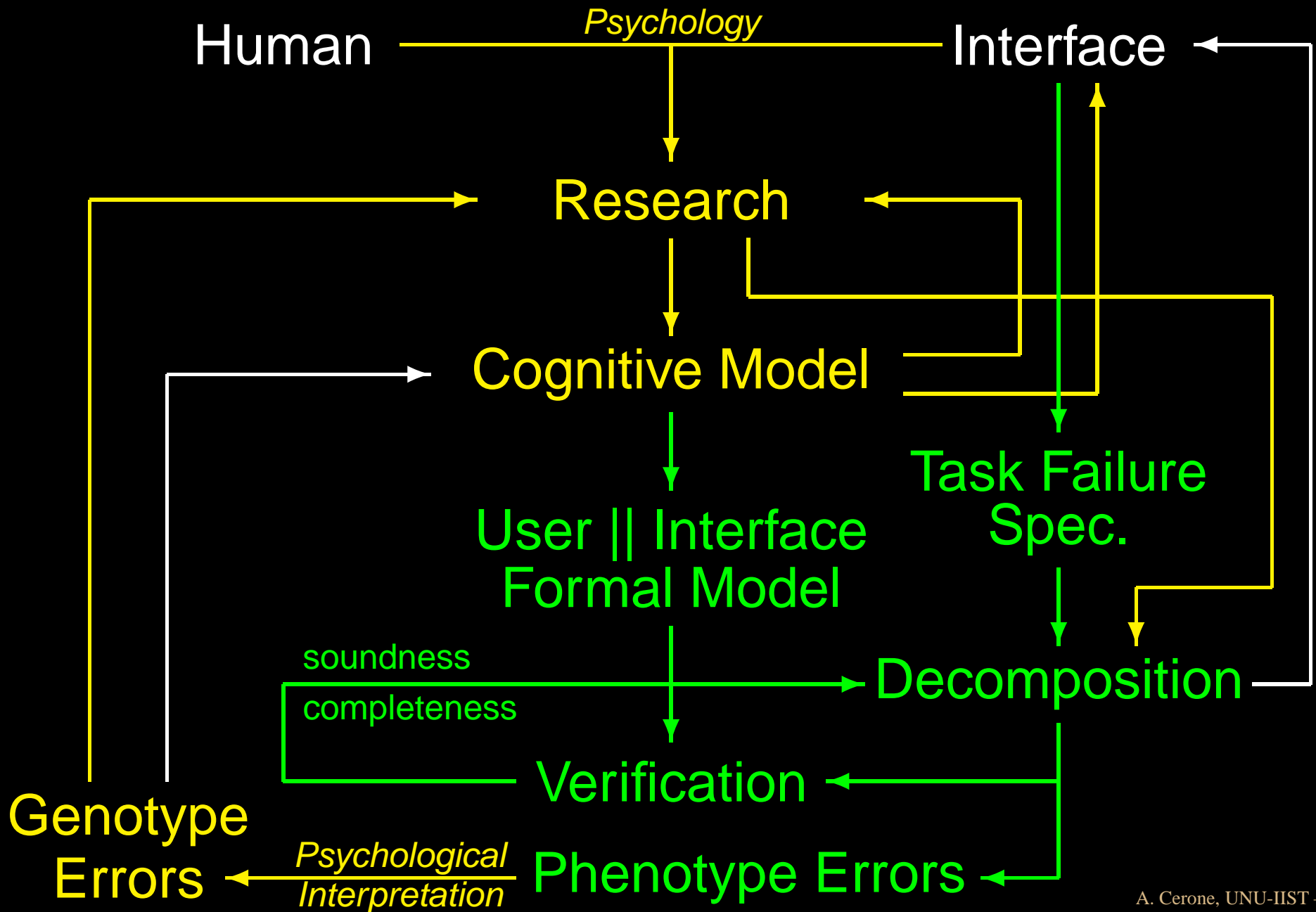












Operator Choice Model

Scanning: The operator searches the interface for a certain property.

Operator Choice Model

Scanning: The operator searches the interface for a certain property.

Identification: The operator identifies part of the interface that may represent the property.

Operator Choice Model

Scanning: The operator searches the interface for a certain property.

Identification: The operator identifies part of the interface that may represent the property.

Classification: The operator

- assesses whether the property is in need of further interest;
- if so, gives some form of priority to the property.

Operator Choice Model

Scanning: The operator searches the interface for a certain property.

Identification: The operator identifies part of the interface that may represent the property.

Classification: The operator

- assesses whether the property is in need of further interest;
- if so, gives some form of priority to the property.

Decision on how to resolve the situation.

Operator Choice Model

Scanning: The operator searches the interface for a certain property.

Identification: The operator identifies part of the interface that may represent the property.

Classification: The operator

- assesses whether the property is in need of further interest;
- if so, gives some form of priority to the property.

Decision on how to resolve the situation.

Action to be performed as a series of interaction with the interface.

OCM for Nuclear Plant

Scanning: The operator scans among each of the individual reactor readouts on the interface **searching for any anomalies**.

Identification: The operator identifies a particular readout.

Classification: The operator

- assesses whether the identified readout describes a **normal** or **abnormal** operation of the plant;
- if abnormal, gives a priority to the operation according to its urgency to be resolved.

Decision on how to **resolve the abnormal situation**.

Action to be performed as a series of interaction with the interface and with internal and/or external authorities.

OCM for Air Traffic Control

Scanning: The operator scans among each pair of aircraft searching for a pair that may violate separation.

Identification: The operator identifies a pair of aircraft.

Classification: The operator

- assesses whether the identified pair of aircraft will eventually violate separation (**in conflict**) or not (**not in conflict**);
- if so, gives a priority to the conflict according to its urgency to be resolved.

Decision on how to **resolve the conflict**.

Action to be performed as a series of interaction with the interface.

Air Traffic Control (ATC)

- Aircraft fly along straight-line segments — called *flight paths* — between *waypoints* within a fixed sector of airspace.

Air Traffic Control (ATC)

- Aircraft fly along straight-line segments — called *flight paths* — between *waypoints* within a fixed sector of airspace.
- Aircraft *horizontal separation* must be at least **5 miles**.

Air Traffic Control (ATC)

- Aircraft fly along straight-line segments — called *flight paths* — between *waypoints* within a fixed sector of airspace.
- Aircraft *horizontal separation* must be at least **5 miles**.
- A *pair* of aircraft *violate separation* when the horizontal distance between them is **less than 5 miles** (*separation violation*).

Air Traffic Control (ATC)

- Aircraft fly along straight-line segments — called *flight paths* — between *waypoints* within a fixed sector of airspace.
- Aircraft *horizontal separation* must be at least **5 miles**.
- A *pair* of aircraft *violate separation* when the horizontal distance between them is **less than 5 miles** (*separation violation*).
- A *pair* of aircraft is in *conflict* when their pathways are such that the two aircraft will **eventually violate separation**.

ATC Simulator

- The ATC operator's task involves monitoring the movement of aircraft on a screen, looking for pair of aircraft that *may violate separation*.

ATC Simulator

- The ATC operator's task involves monitoring the movement of aircraft on a screen, looking for pair of aircraft that *may violate separation*.
- When such a conflict is detected, the operator uses a mouse to select one of the aircraft and change its speed using a pulldown menu.

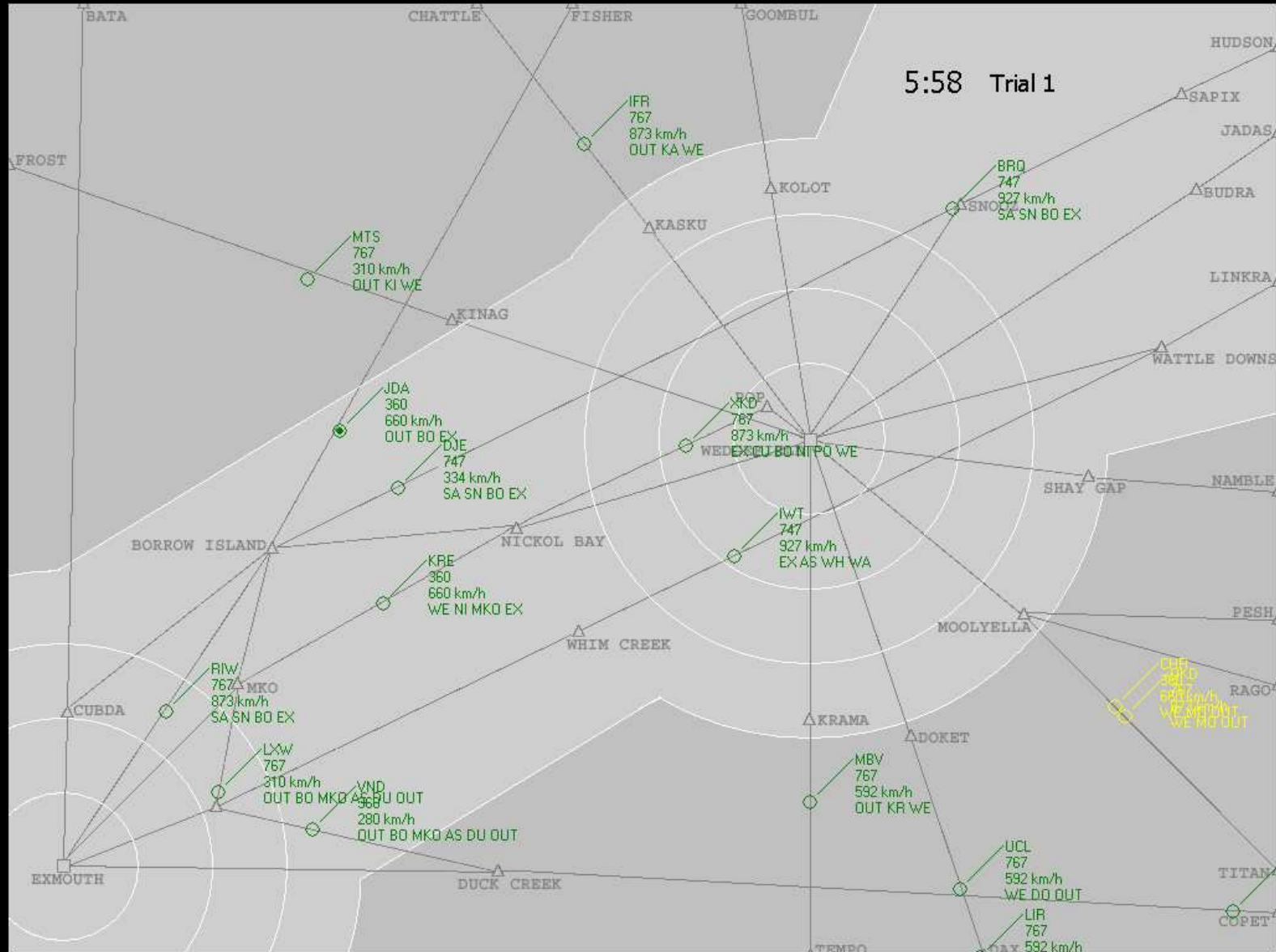
ATC Simulator

- The ATC operator's task involves monitoring the movement of aircraft on a screen, looking for pair of aircraft that *may violate separation*.
- When such a conflict is detected, the operator uses a mouse to select one of the aircraft and change its speed using a pulldown menu.
- The *goal of the task* is to *resolve all conflicts* before they violate separation, while *not introducing any new conflict*.

ATC Simulator

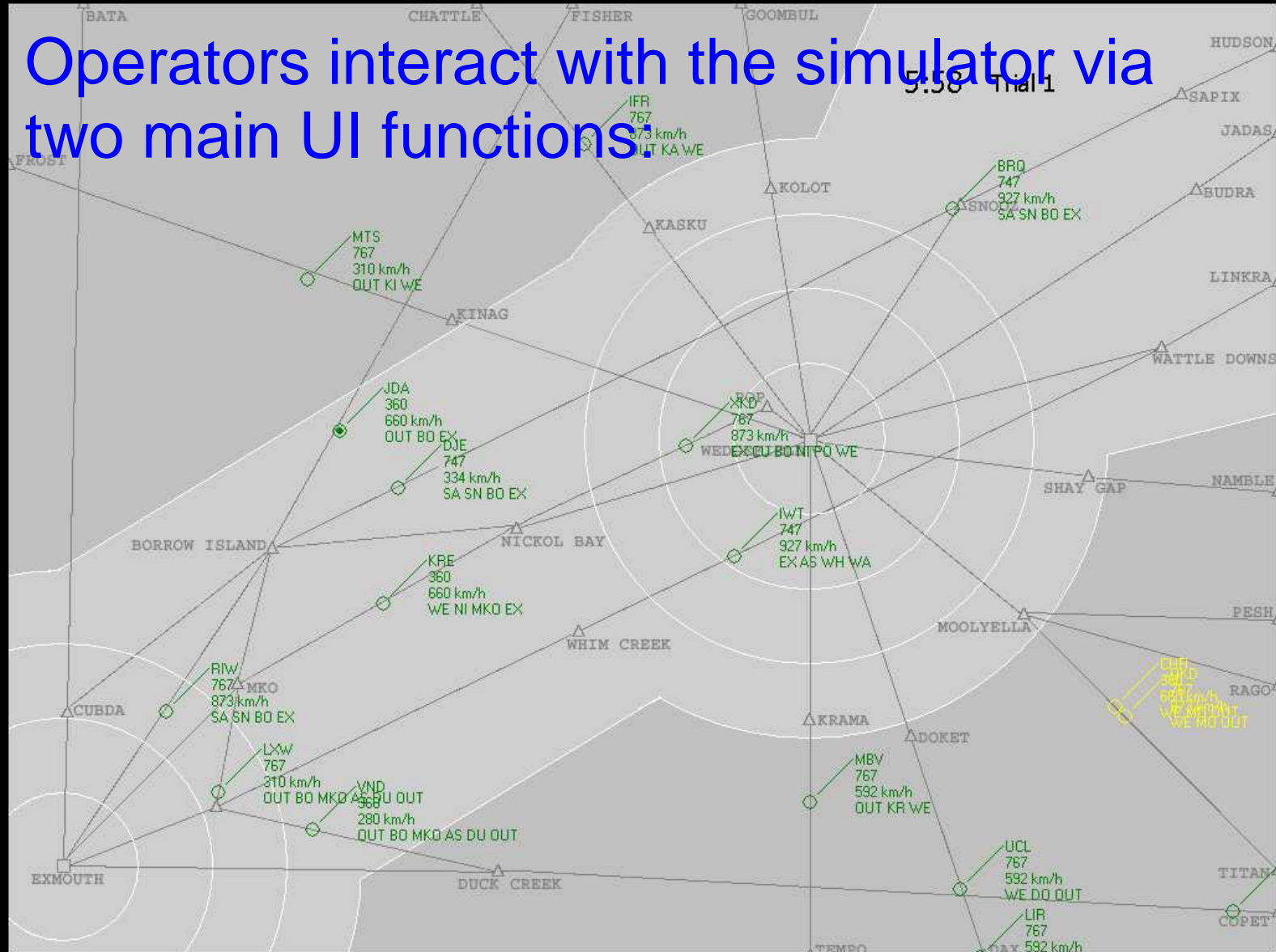
- The ATC operator's task involves monitoring the movement of aircraft on a screen, looking for pair of aircraft that *may violate separation*.
- When such a conflict is detected, the operator uses a mouse to select one of the aircraft and change its speed using a pulldown menu.
- The **goal of the task** is to **resolve all conflicts** before they violate separation, while **not introducing any new conflict**.
- We have a **task failure** when **separation is violated**.

ATC Simulator Screenshot



ATC Simulator Screenshot

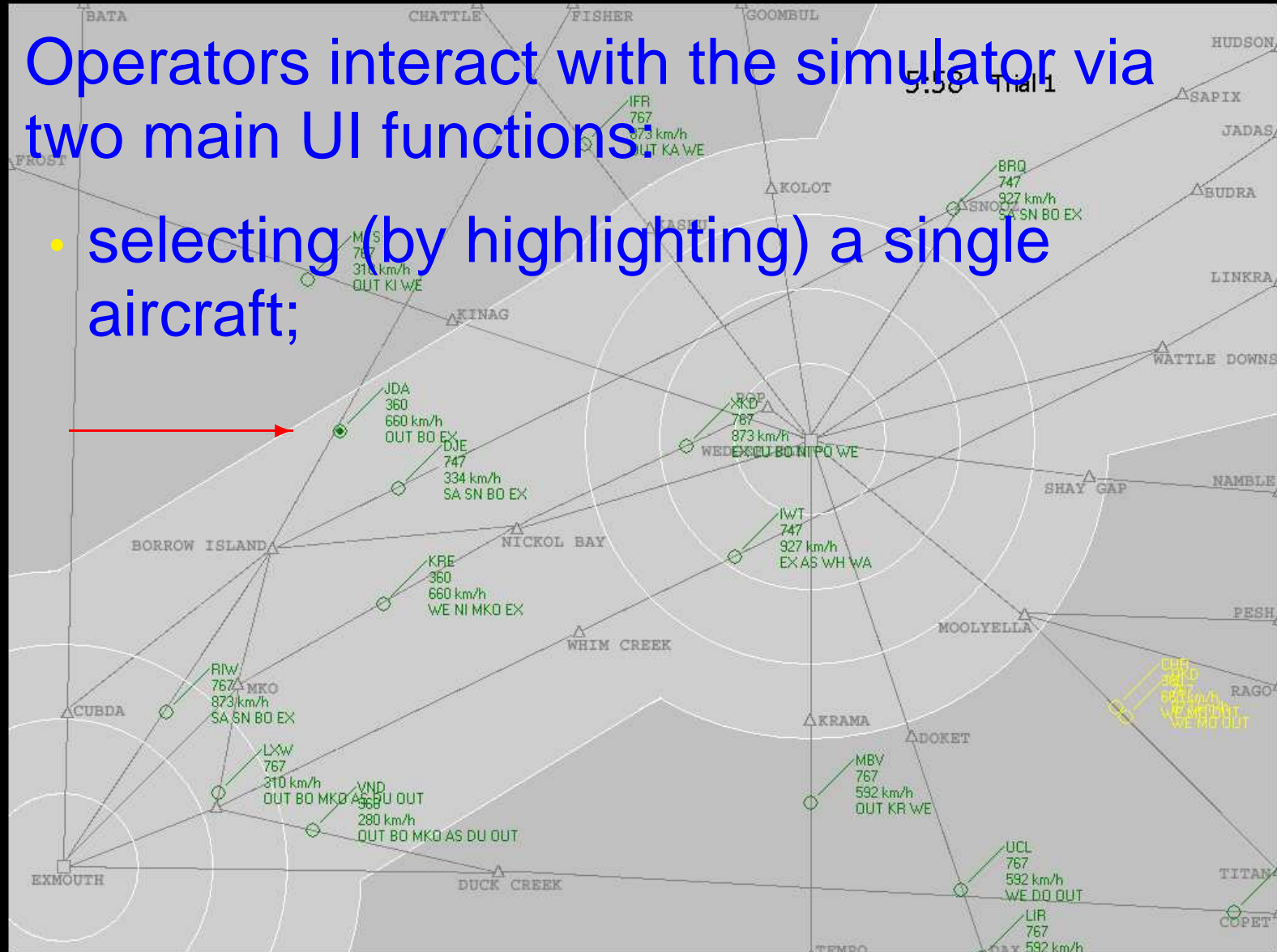
Operators interact with the simulator via two main UI functions:



ATC Simulator Screenshot

Operators interact with the simulator via two main UI functions.

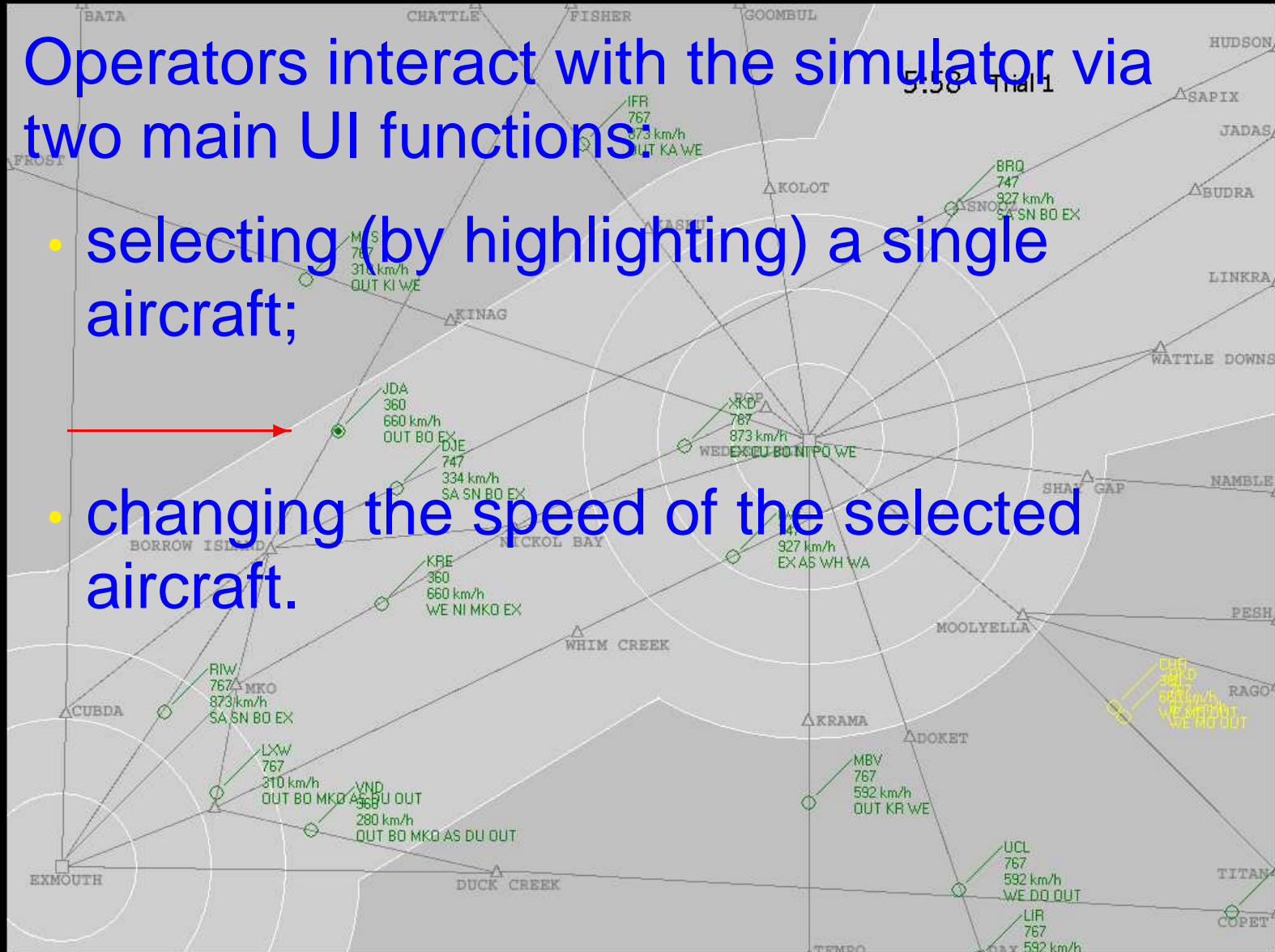
- selecting (by highlighting) a single aircraft;



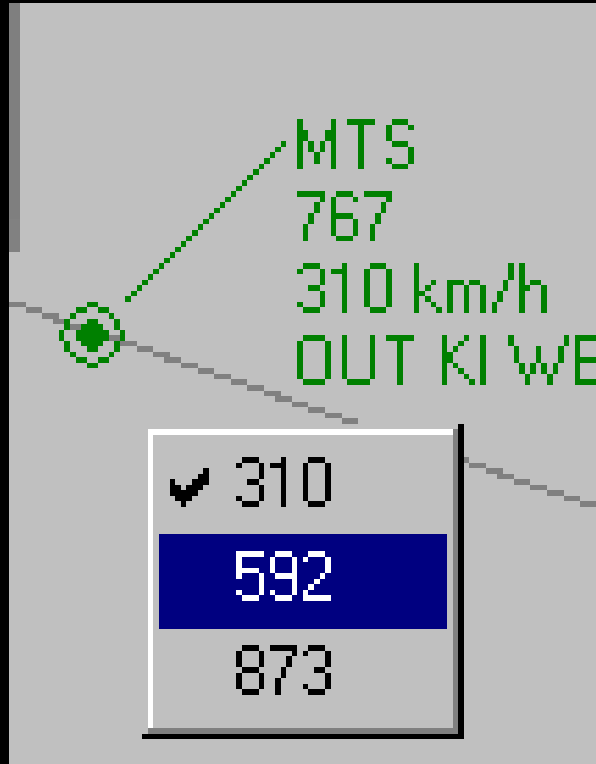
ATC Simulator Screenshot

Operators interact with the simulator via two main UI functions:

- selecting (by highlighting) a single aircraft;
- changing the speed of the selected aircraft.



Speed Menu

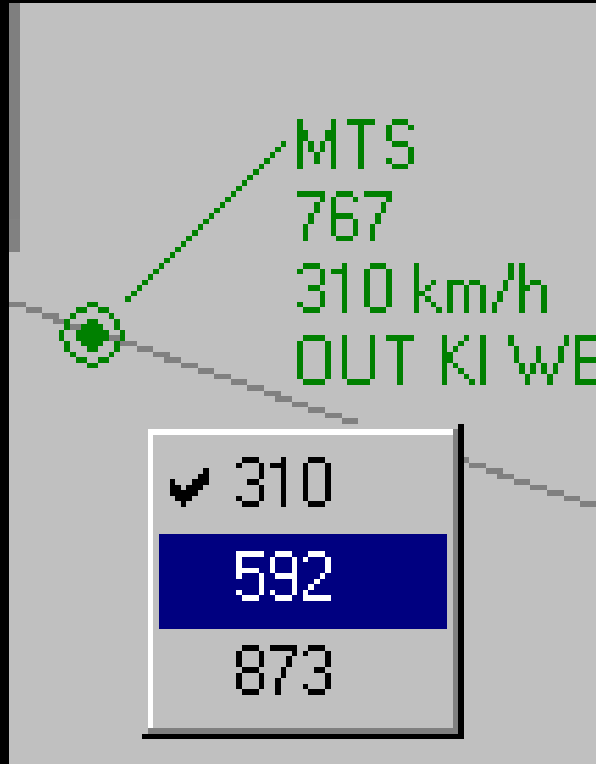


Speed Menu



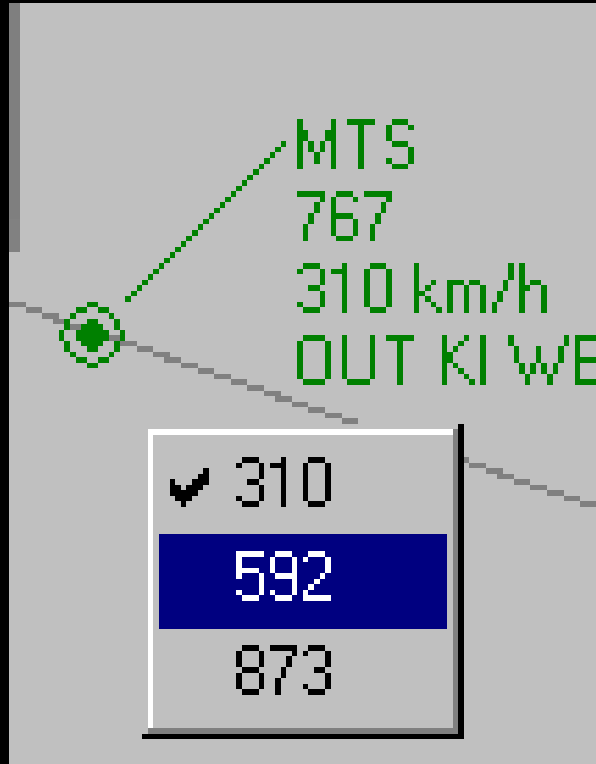
Open the menu by clicking the right button.

Speed Menu



Open the menu by clicking the right button.
The menu appears at the position of the cursor.

Speed Menu



Open the menu by clicking the right button.
The menu appears at the position of the cursor.
Selected the speed by left clicking on the desired menu entry.

Operator Erros

- **slip:** inadvertently select a wrong or the current speed

Operator Erros

- **slip**: inadvertently select a wrong or the current speed
 \Leftarrow selction task closure (cognitive problem)

Operator Errors

- **slip**: inadvertently select a wrong or the current speed
 \Leftarrow **selection task closure (cognitive problem)**
- **mistaken identity**: change the speed of an aircraft different from the intended one

Operator Errors

- **slip**: inadvertently select a wrong or the current speed
 \Leftarrow selection task closure (cognitive problem)
- **mistaken identity**: change the speed of an aircraft different from the intended one
 \Leftarrow the menu appears at the position of the cursor (usability problem)

Operator Errors

- **slip**: inadvertently select a wrong or the current speed
 \Leftarrow selection task closure (cognitive problem)
- **mistaken identity**: change the speed of an aircraft different from the intended one
 \Leftarrow the menu appears at the position of the cursor (usability problem)
- **mis-classification, mis-prioritization, conflict generation**

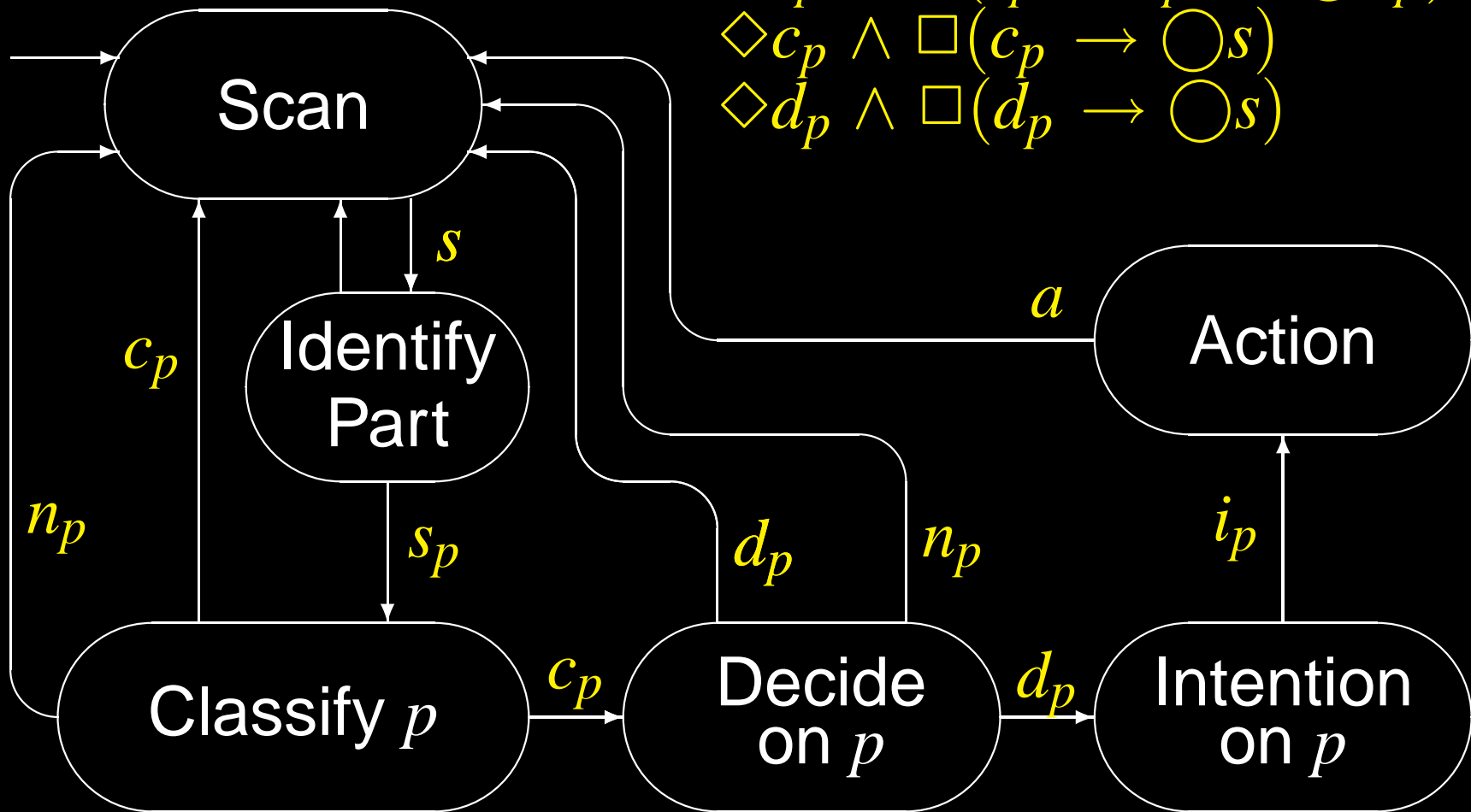
Operator Errors

- **slip**: inadvertently select a wrong or the current speed
 \Leftarrow **selection task closure (cognitive problem)**
- **mistaken identity**: change the speed of an aircraft different from the intended one
 \Leftarrow **the menu appears at the position of the cursor (usability problem)**
- **mis-classification, mis-prioritization, conflict generation**

The operator can **recover** from these **errors** without causing **separation violation (task failure)**

Task Failure Decomposition

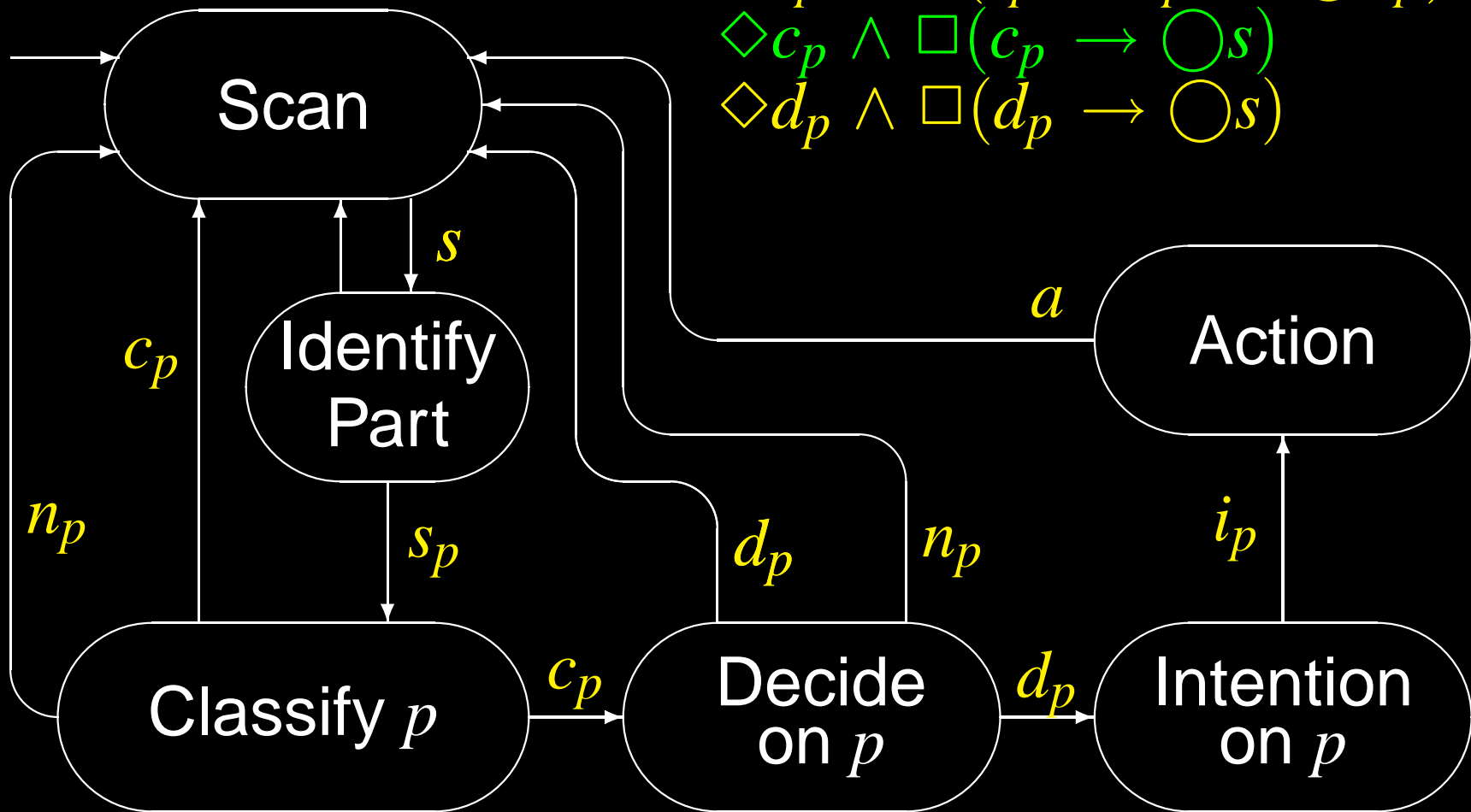
$\Box \neg i_p$ is decomposed as $\Box \neg s_p$

$$\begin{aligned} & \Diamond s_p \wedge \Box (s_p \vee c_p \rightarrow \bigcirc n_p) \\ & \Diamond c_p \wedge \Box (c_p \rightarrow \bigcirc s) \\ & \Diamond d_p \wedge \Box (d_p \rightarrow \bigcirc s) \end{aligned}$$


Task Failure Decomposition

$\square \neg i_p$ is decomposed as $\square \neg s_p$

$\diamond s_p \wedge \square (s_p \vee c_p \rightarrow \bigcirc n_p)$
 $\diamond c_p \wedge \square (c_p \rightarrow \bigcirc s)$
 $\diamond d_p \wedge \square (d_p \rightarrow \bigcirc s)$



Single Mis-prioritisation

$$c_p \rightarrow \bigcirc s$$

(phenotype error)

Single Mis-prioritisation

$$c_p \rightarrow \bigcirc s$$

(phenotype error)

Possible **genotype errors** are

- mis-calculation
- mis-storage
- mis-retrival

of the time planned for corrective actions

Single Mis-prioritisation

$$c_p \rightarrow \bigcirc s$$

(phenotype error)

Possible **genotype errors** are

- mis-calculation
- mis-storage
- mis-retrival

of the time planned for corrective actions

\Rightarrow possible **recovery**

(through new calculation at a next scan)

Persistent Mis-prioritisation

$$\Diamond c_p \wedge \Box(c_p \rightarrow \bigcirc s)$$

(phenotype error)

Persistent Mis-prioritisation

$$\Diamond c_p \wedge \Box(c_p \rightarrow \bigcirc s)$$

(phenotype error)

Possible **genotype errors**

- **perception distorted** \implies memory of result of previous **mis-calculation** keeps emerging

due to

- **distraction**
- **similarity with observed non-conflicts**
- **high workload**

ATC: References

Peter Lindsay and Simon Connelly. **Modelling Erroneous Operator Behaviours for an Air-Traffic Control Task**. AUIC 2002.

ATC: References

Peter Lindsay and Simon Connelly. **Modelling Erroneous Operator Behaviours for an Air-Traffic Control Task.** AUIC 2002.

Antonio Cerone, Peter Lindsay and Simon Connelly. **Formal Analysis of Human-computer Interaction using Model-checking.** SEFM 2005.

Antonio Cerone, Simon Connelly and Peter Lindsay. **Formal Analysis of Human Operator Behavioural Patterns in Interactive Surveillance Systems.** Software and System Modeling 7(3), Springer, pages 273–286, 2008.

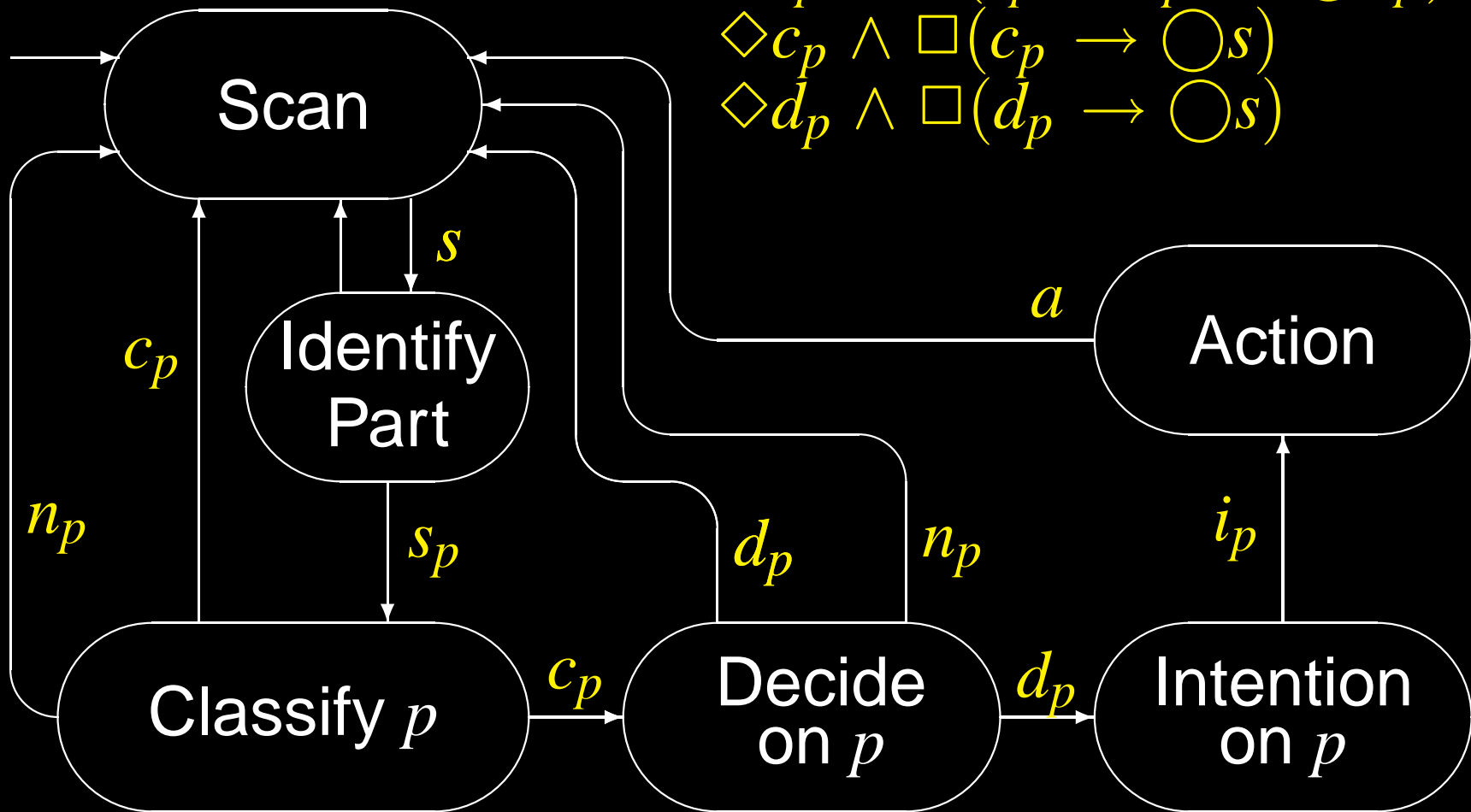
Task Failure Decomposition

$\Box \neg i_p$ is decomposed as $\Box \neg s_p$

$$\Diamond s_p \wedge \Box (s_p \vee c_p \rightarrow \bigcirc n_p)$$

$$\Diamond c_p \wedge \Box (c_p \rightarrow \bigcirc s)$$

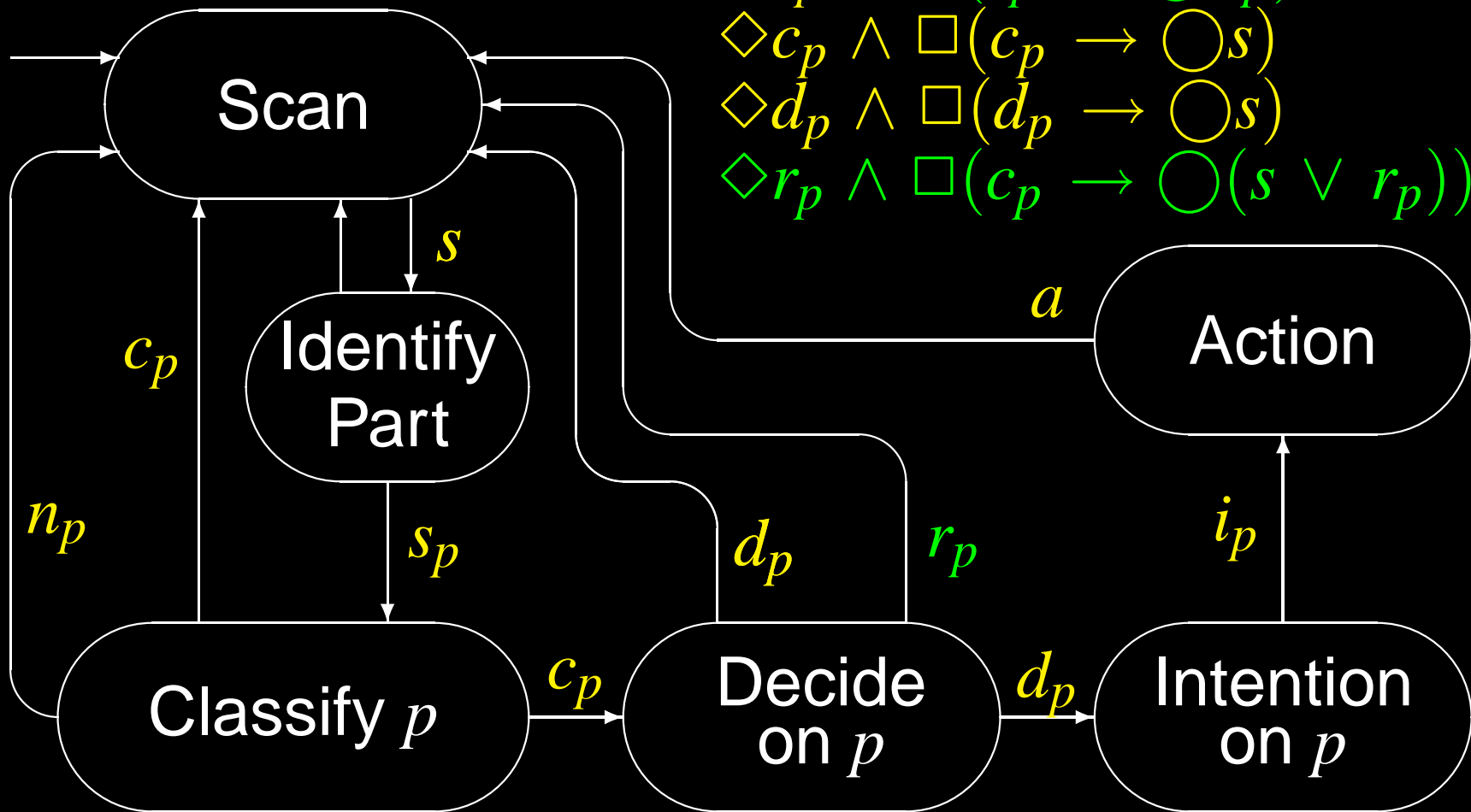
$$\Diamond d_p \wedge \Box (d_p \rightarrow \bigcirc s)$$



Complete Decomposition

$\Box \neg i_p$ is decomposed as

$$\begin{aligned} & \Box \neg s_p \\ & \Diamond s_p \wedge \Box(s_p \rightarrow \bigcirc n_p) \\ & \Diamond c_p \wedge \Box(c_p \rightarrow \bigcirc s) \\ & \Diamond d_p \wedge \Box(d_p \rightarrow \bigcirc s) \\ & \Diamond r_p \wedge \Box(c_p \rightarrow \bigcirc(s \vee r_p)) \end{aligned}$$



Contrary Decision Process

$$\Diamond r_p \wedge \Box(c_p \rightarrow \bigcirc(s \vee r_p))$$

(phenotype error)

Possible **genotype error** is

- memory of previous decisions on similar pairs resulting in **unnecessary actions**

due to

- **fear**
- **high workload**

Error Cause

What caused such an error?

Error Cause

What caused such an error?

- use of the same action name *n* to denote the results of two cognitive processes

Error Cause

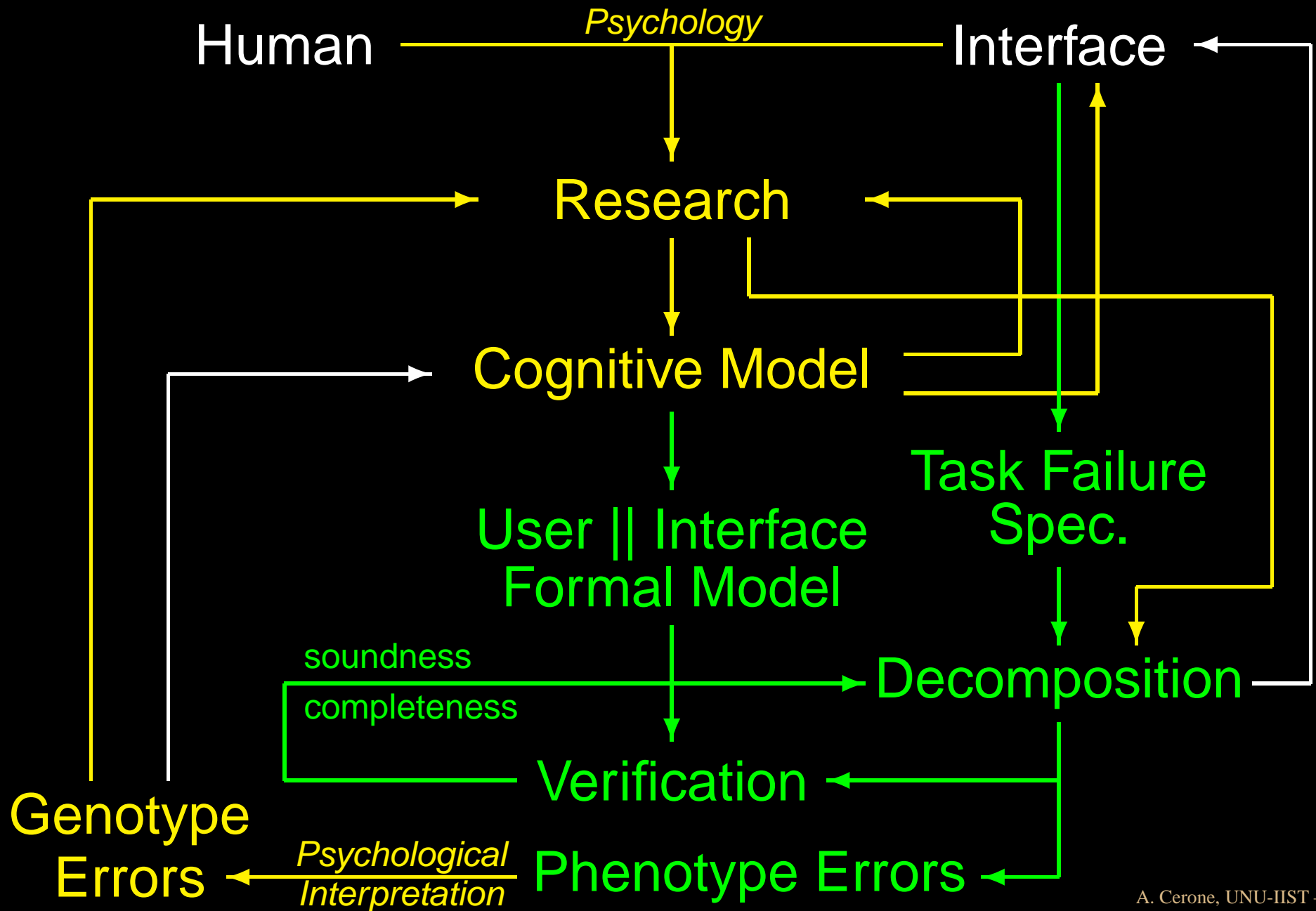
What caused such an error?

- use of the same action name *n* to denote the results of two cognitive processes
- aim at an elegant and easy to understand (to psychologists) formal model

Error Cause

What caused such an error?

- use of the same action name *n* to denote the results of two cognitive processes
- aim at an elegant and easy to understand (to psychologists) formal model
⇒ focus on syntactical look of formulae rather than on their interpretation on the model

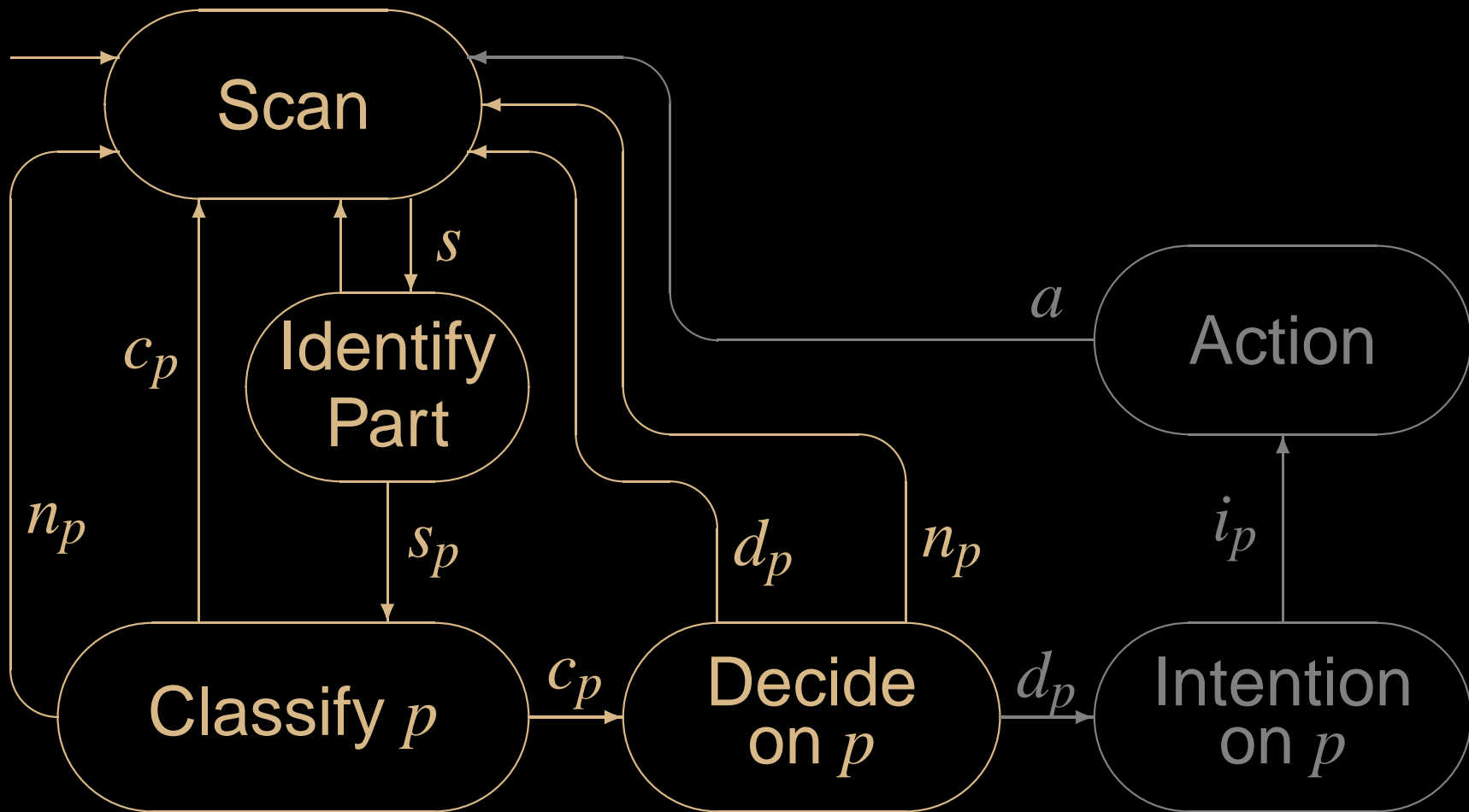


Exercise: Counterexample?

Find and analyse the counterexample
which falsifies completeness

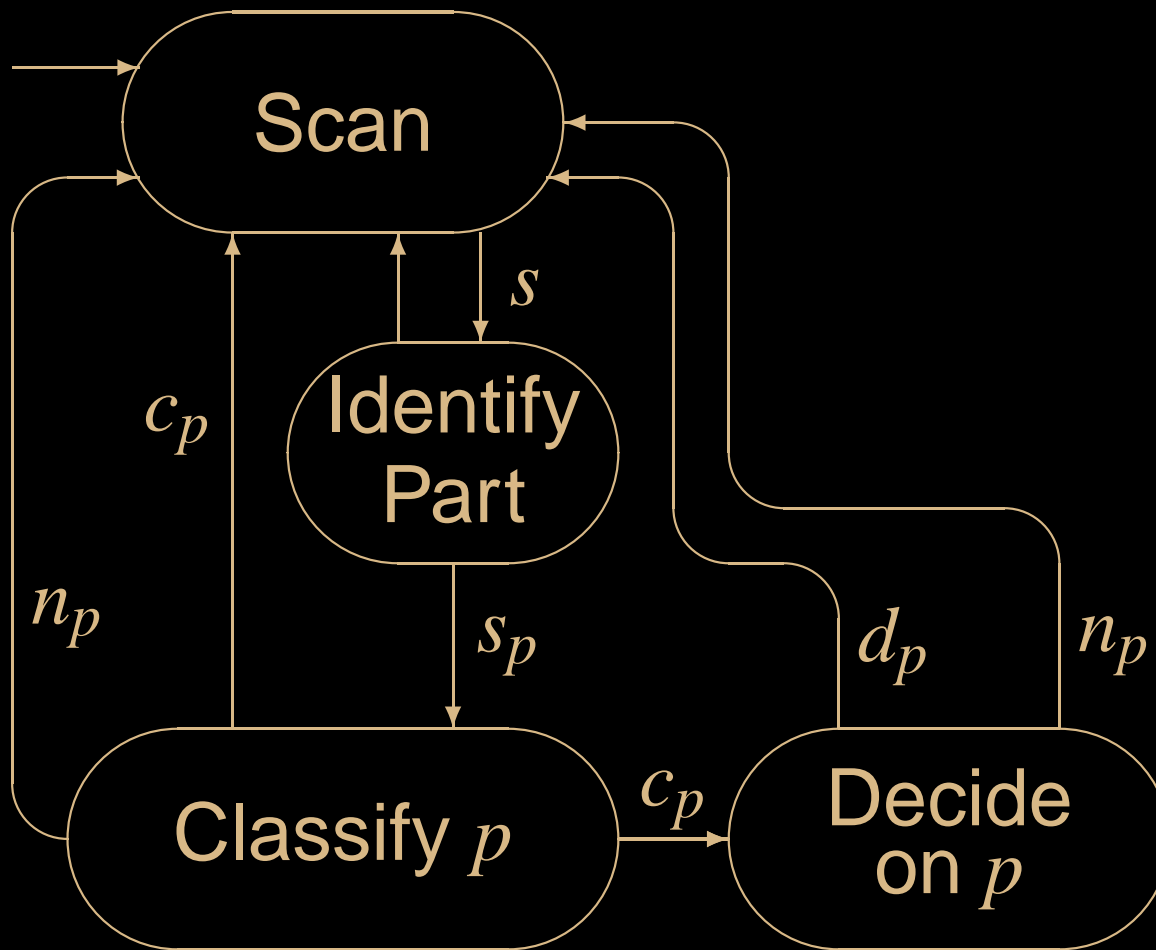
Exercise: Counterexample?

Find and analyse the counterexample



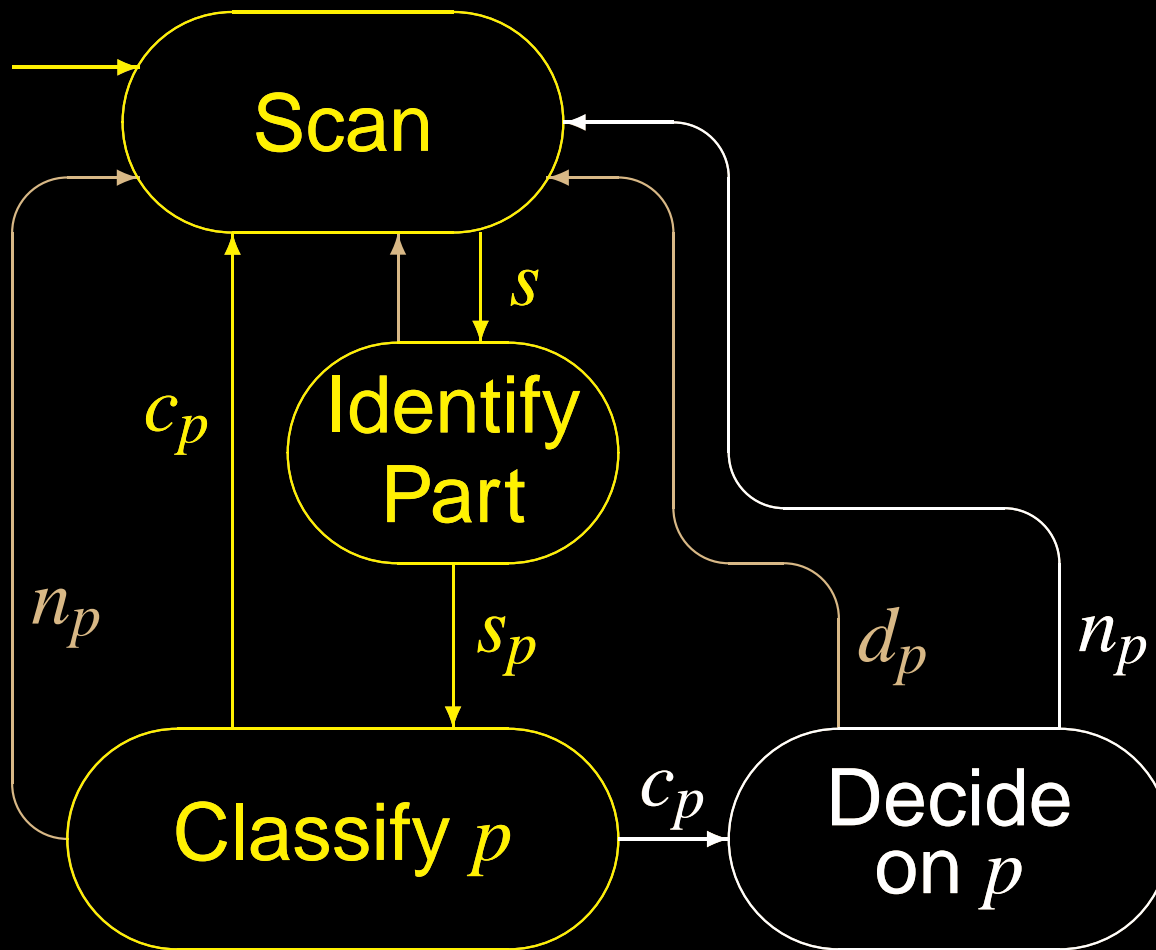
Exercise: Counterexample?

Find and analyse the counterexample

$$s \longrightarrow s_p \longrightarrow c_p \longrightarrow s \longrightarrow s_p \longrightarrow c_p \longrightarrow n_p \longrightarrow s \longrightarrow \dots$$


Exercise: Counterexample?

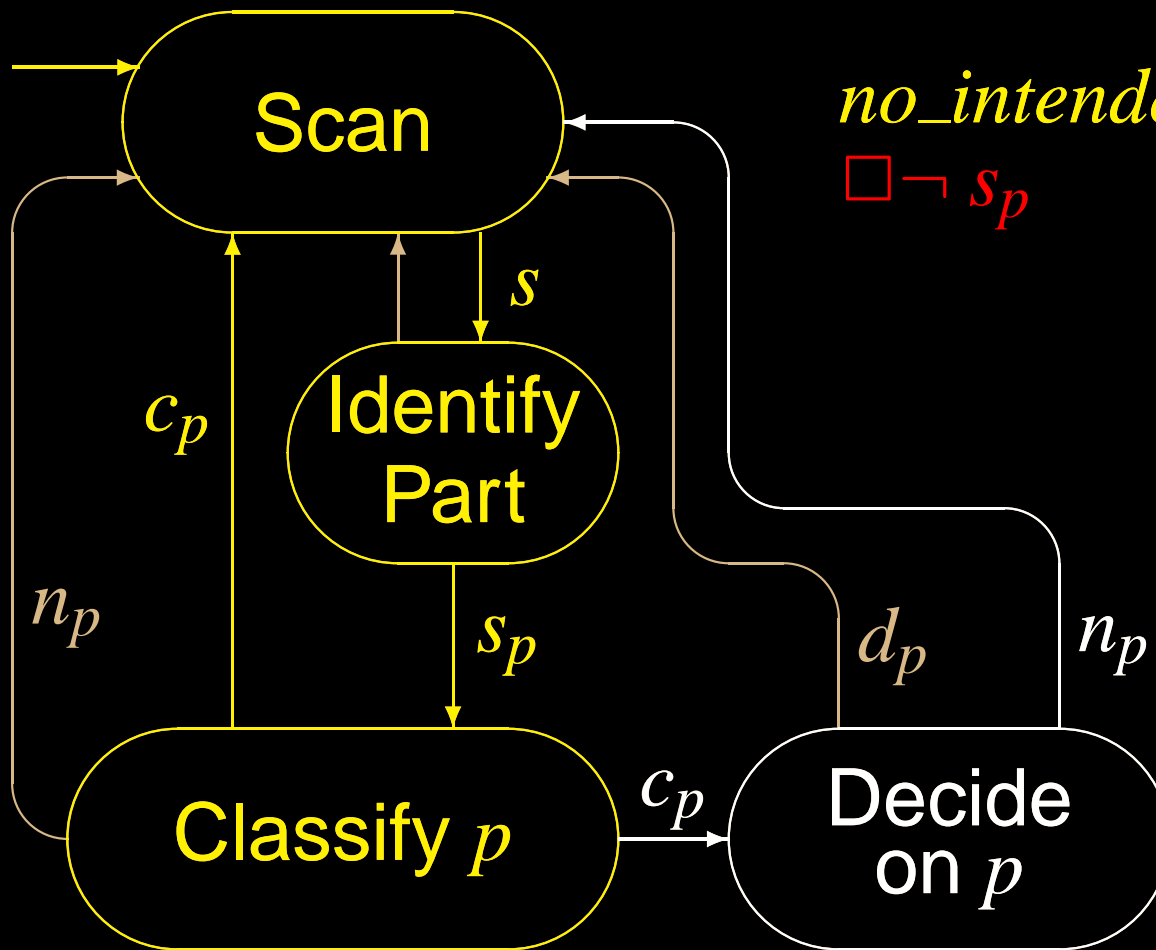
Find and analyse the counterexample

$$s \longrightarrow s_p \longrightarrow c_p \longrightarrow s \longrightarrow s_p \longrightarrow c_p \longrightarrow n_p \longrightarrow s \longrightarrow \dots$$


Exercise: Counterexample?

Find and analyse the counterexample

$s \longrightarrow \textcircled{s_p} \longrightarrow c_p \longrightarrow s \longrightarrow s_p \longrightarrow c_p \longrightarrow n_p \longrightarrow s \longrightarrow \dots$

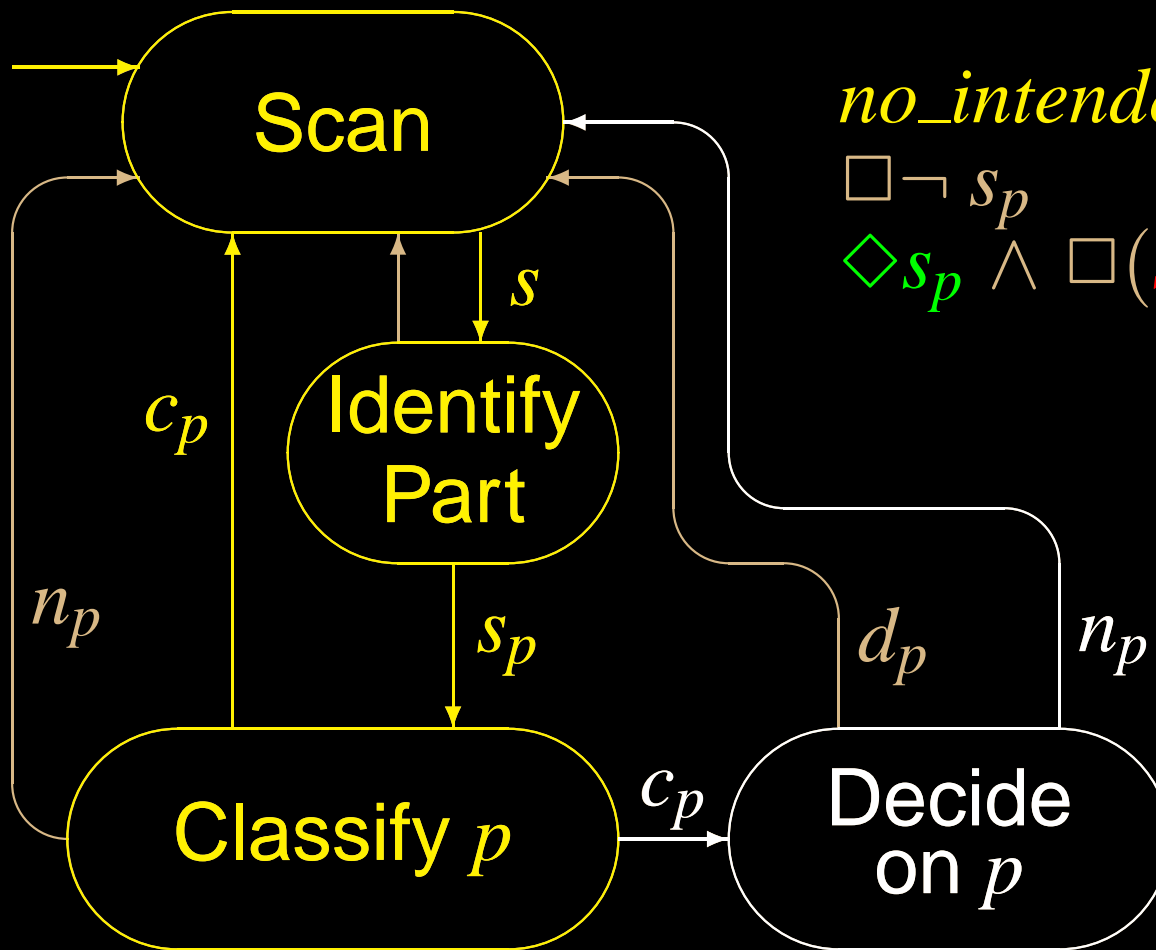


$no_intended_response_p :$
 $\square \neg s_p$

Exercise: Counterexample?

Find and analyse the counterexample

$s \longrightarrow \textcircled{s_p} \longrightarrow c_p \longrightarrow s \longrightarrow s_p \longrightarrow c_p \longrightarrow n_p \longrightarrow s \longrightarrow \dots$



$no_intended_response_p :$

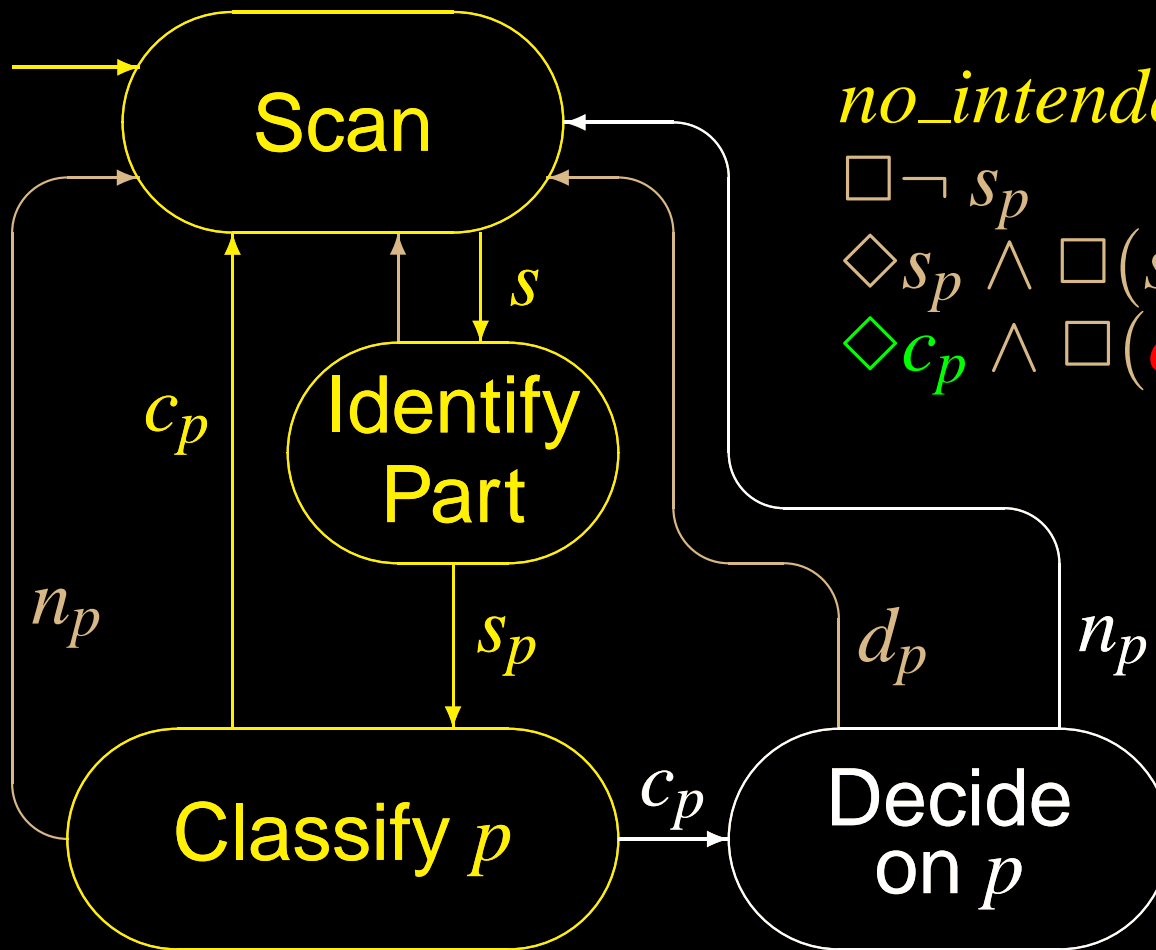
$\Box \neg s_p$

$\Diamond s_p \wedge \Box (s_p \vee c_p \rightarrow \bigcirc n_p)$

Exercise: Counterexample?

Find and analyse the counterexample

$s \longrightarrow s_p \longrightarrow c_p \longrightarrow s \longrightarrow s_p \longrightarrow c_p \longrightarrow n_p \longrightarrow s \longrightarrow \dots$



no_intended_response_p :

$\Box \neg s_p$

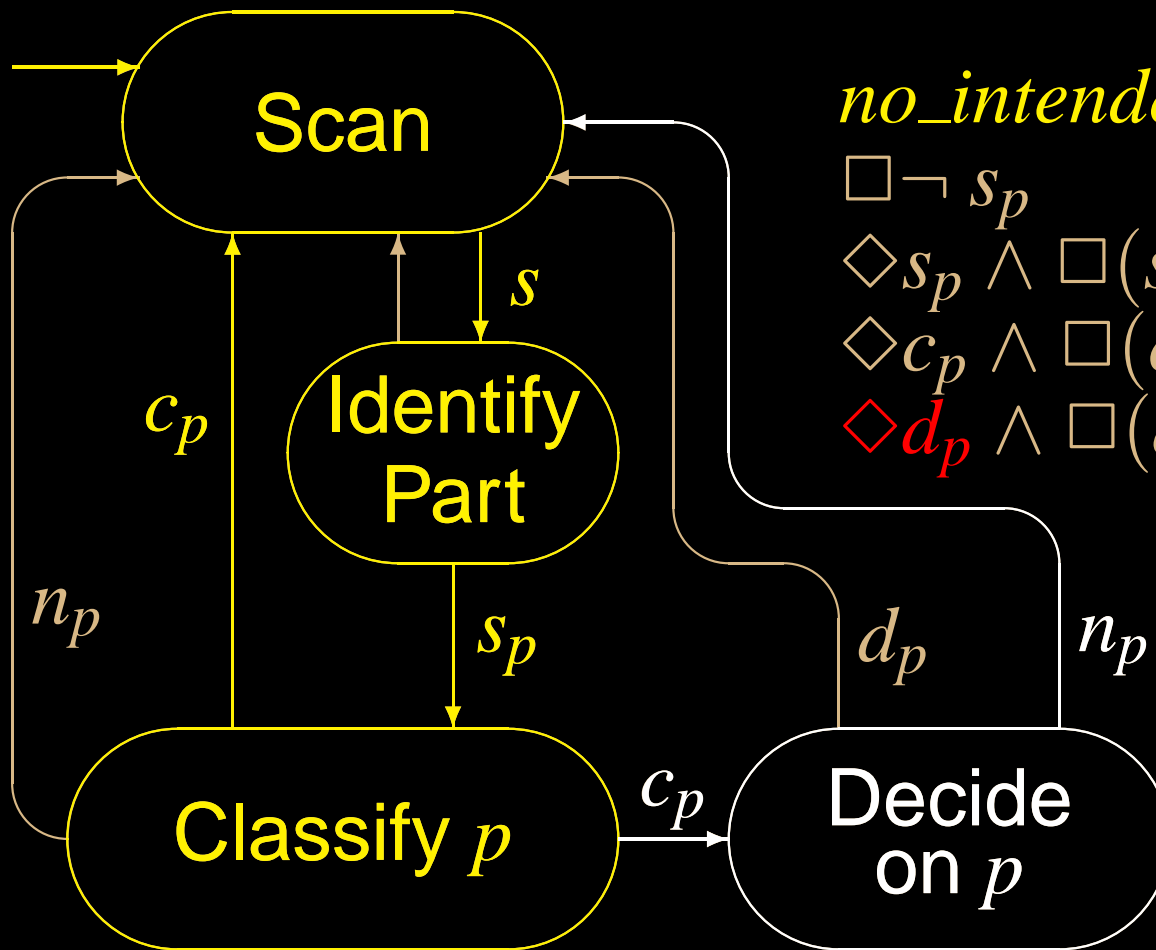
$\Diamond s_p \wedge \Box (s_p \vee c_p \rightarrow \bigcirc n_p)$

$\Diamond c_p \wedge \Box (c_p \rightarrow \bigcirc s)$

Exercise: Counterexample?

Find and analyse the counterexample

$s \longrightarrow s_p \longrightarrow c_p \longrightarrow s \longrightarrow s_p \longrightarrow c_p \longrightarrow n_p \longrightarrow s \longrightarrow \dots$



no_intended_response_p :

$\Box \neg s_p$

$\Diamond s_p \wedge \Box (s_p \vee c_p \rightarrow \bigcirc n_p)$

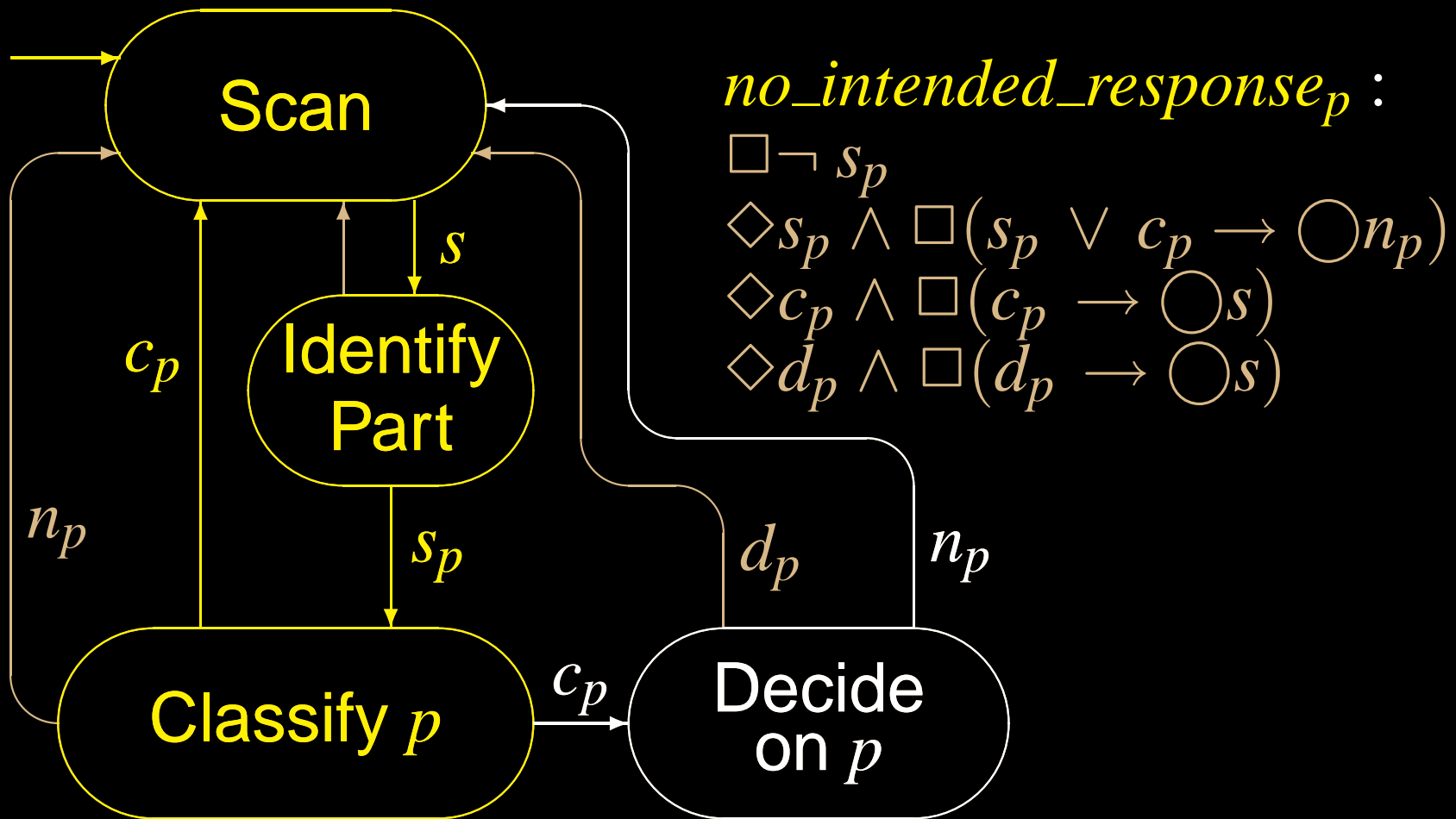
$\Diamond c_p \wedge \Box (c_p \rightarrow \bigcirc s)$

$\Diamond d_p \wedge \Box (d_p \rightarrow \bigcirc s)$

Exercise: Counterexample?

Find and analyse the counterexample

$s \longrightarrow s_p \longrightarrow c_p \longrightarrow s \longrightarrow s_p \longrightarrow c_p \longrightarrow n_p \longrightarrow s \longrightarrow \dots$



References

[Dix et al. 98]

Alan Dix, Janet Finlay, Gregory Abowd, Russel Beale.

Human-Computer Interaction.

Prentice Hall, 2nd Edition, 1998.

HCI Textbook

One of the most complete general textbooks in HCI, also introduces the use of several formal notations, such as Petri nets, CSP, temporal logic, Z. There is now a 3rd edition.

Complementary materials available online at

<http://www.hiraeth.com/books/hci/>

[Preece et al. 94]

Jenny Preece, Yvonne Rogers, Helen Sharp,
David Benyon, Simon Holland and Tom Carey.

Human-Computer Interaction.

Addison Wesley, 1994.

HCI Textbook

The first HCI textbook to contain all pedagogical features (examples, exercises, etc.). Now a bit old. Book review available online at

<http://www.acm.org/~perlman/preece.html>

[Card et al. 83]

Stuart K. Card, Thomas P. Moran and Allen Newell.

The Psychology of Human-Computer interaction.
Lawrence Erlbaum Associates, 1983.

HCI General Book

Classical book that defines the early theoretical basis of HCI from an Information Processing perspective. Develops and describes the **Model Human Processor** in details.

[Dix 91]

Alan Dix.

Formal Methods for Interactive Systems.

Academic Press, 1991.

FMIS Textbook

Out of print, but available online at

<http://www.hiraeth.com/books/formal/>

[FMIS 06]

A. Cerone and P. Curzon.

Proceedings of FMIS 2006.

ENTCS 183, Elsevier, 2007

Extended version of a selection of the papers
has been published in

Software and System Modeling Vol. 4, No. 2,
Springer, 2008

[FMIS 07]

A. Cerone and P. Curzon.

Proceedings of FMIS 2007.

ENTCS , Elsevier, 2007

Extended version of a selection of the papers
has been published in

Formal Aspects of Computing Vol. 21, No. 6,
Springer, 2009

[Cerone and Elgegbyan 07]

A. Cerone and N. Elgegbyan.

Model-checking Driven Design of Interactive Systems.

ENTCS 183, Elsevier, 2007, pages 3–20.

Formal Methods Paper

Use of **model-checking** to improve the interface design with respect to **security properties**.

[Lindsay et. al. 02]

P. Lindsay and S. Connelly.

Modelling Erroneous Operator Behaviour for an Air-traffic Control Task.

AUIC 2002, Conferences in Research and Practice in Information Technology, Vol. 7, Australian Computer Society, pages 43–54.

Formal Methods Paper

Incorrect decomposition for the ATC Example.

[Cerone et al. 05 and 08]

A. Cerone, P. Lindsay and S. Connelly.

Formal Analysis of Human-computer Interaction using Model-checking.

SEFM 2005, IEEE Comp. Soc., 2005, pages 352–361.

A. Cerone, S. Connelly and P. Lindsay.

Formal Analysis of Human Operator Behavioural Patterns in Interactive Surveillance Systems.

Software and Systems Modeling, Vol.7, No. 3, Springer, 2008, pages 273–286.

Formal Methods Paper

On the **correct decomposition** for the **ATC** Example, but the second is the most complete.

End