1° Workshop LICIA

31 Oct. 2011

Systems Architecture

Advances & Challenges

Sacha Krakowiak

Université de Grenoble

Systems Architecture

- Architecture

  the art of designing and constructing buildings

  the art of creating and organizing forms towards a function, in the presence of constraints

Systems Architecture

- **Architecture**

  the art of designing and constructing buildings

  the art of creating and organizing forms towards a function, in the presence of constraints

- **Computing Systems Architecture**

  A model that describes the structure and behavior of a system in terms of elements and relationships

  A method for building a system according to given specifications

Christopher Alexander, *Notes on the Synthesis of Form*, Harvard University Press, 1964

Some tools of the architect's trade

- (Meta) principles
  - Generic rules that may take various concrete forms
    - Abstraction (e.g., hierarchy of abstract machines)
    - Separation of concerns
      - (e.g., separation between policy / mechanisms, interface / implementation)
    - Economy (e.g., optimizing the frequent case, end-to-end principle)
      - …

Some tools of the architect's trade

- (Meta) principles

  Generic rules that may take various concrete forms

  Abstraction (e.g., hierarchy of abstract machines)

  Separation of concerns

  (e.g., separation between policy / mechanisms, interface / implementation)

  Economy (e.g., optimizing the frequent case, end-to-end principle)

  …

- Paradigms

  Paradigm: a design, organization scheme, or structure applicable to a wide class of situations, that may serve as an example, in both senses of the term

  an illustration of an approach

  a model to follow

Some paradigms of systems architecture

- Virtualization
  - giving a concrete form to an ideal object
  - multiplexing a real object

Some paradigms of systems architecture

- Virtualization
  - giving a concrete form to an ideal object
  - multiplexing a real object
- Composition and decomposition
  - separating concerns
    - (individual design / assembly)
  - reusing design and implementation efforts
  - facilitating evolution and maintenance

Some paradigms of systems architecture

- Virtualization
  - giving a concrete form to an ideal object
  - multiplexing a real object
- Composition and decomposition
  - separating concerns
    - (individual design / assembly)
  - reusing design and implementation efforts
  - facilitating evolution and maintenance
- Self-adaptation and reflection
  - reacting to change (both expected and unexpected)
  - optimizing quality criteria

Some paradigms of systems architecture

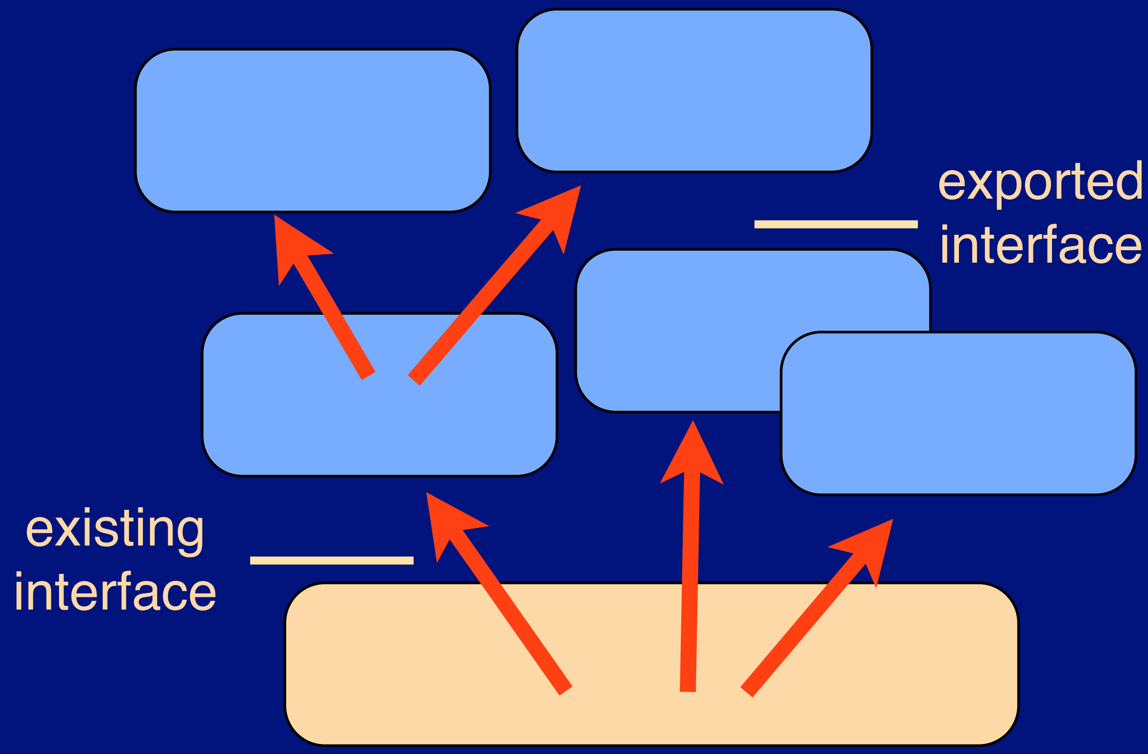
- Virtualization
  - giving a concrete form to an ideal object
  - multiplexing a real object
- Composition and decomposition
  - separating concerns
    - (individual design / assembly)
  - reusing design and implementation efforts
  - facilitating evolution and maintenance
- Self-adaptation and reflection
  - reacting to change (both expected and unexpected)
  - optimizing quality criteria

a “transversal” concern

Formalization, proof

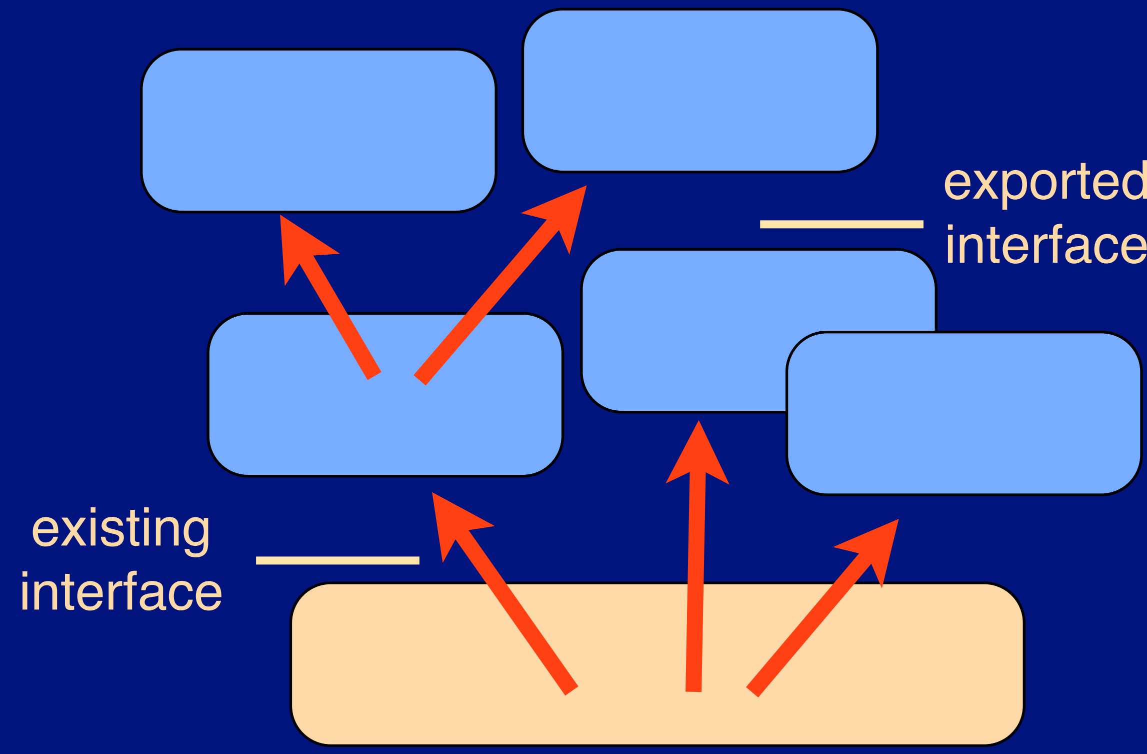
The two faces of virtualization

- Ascending (abstraction)
  - A tool for resource sharing
  - Creating “high level” resources
  - Multiplexing “low level” resources

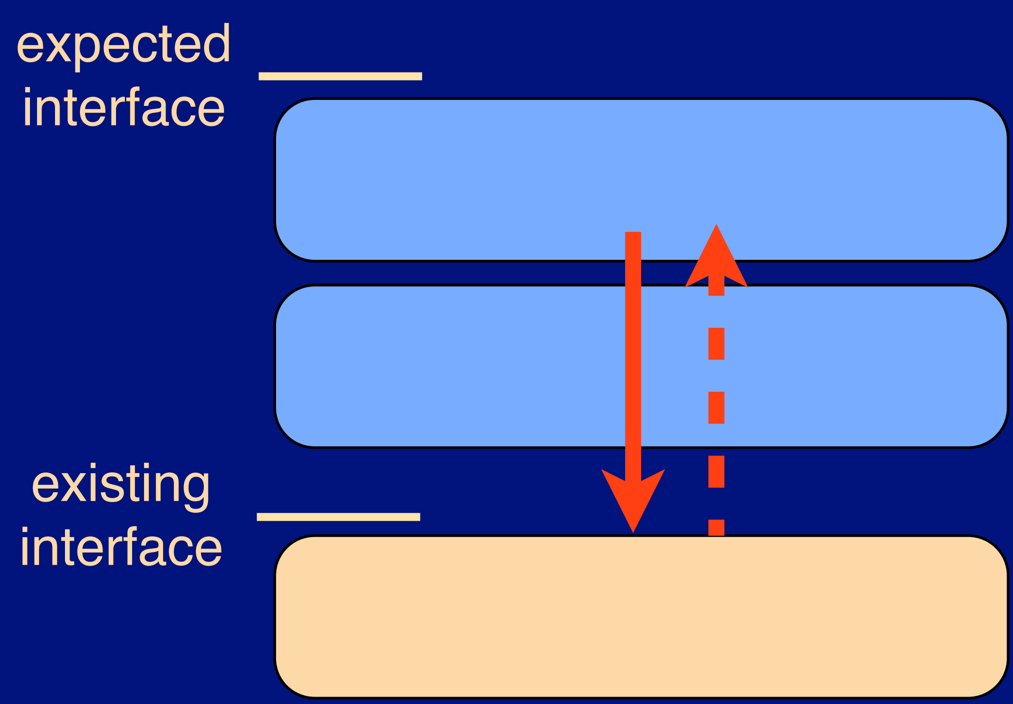


The two faces of virtualization

- Ascending (abstraction)
  - A tool for resource sharing
  - Creating “high level” resources
  - Multiplexing “low level” resources



- Descending (refinement)
  - A design tool
  - From specification to implementation
  - Hierarchy of abstract machines

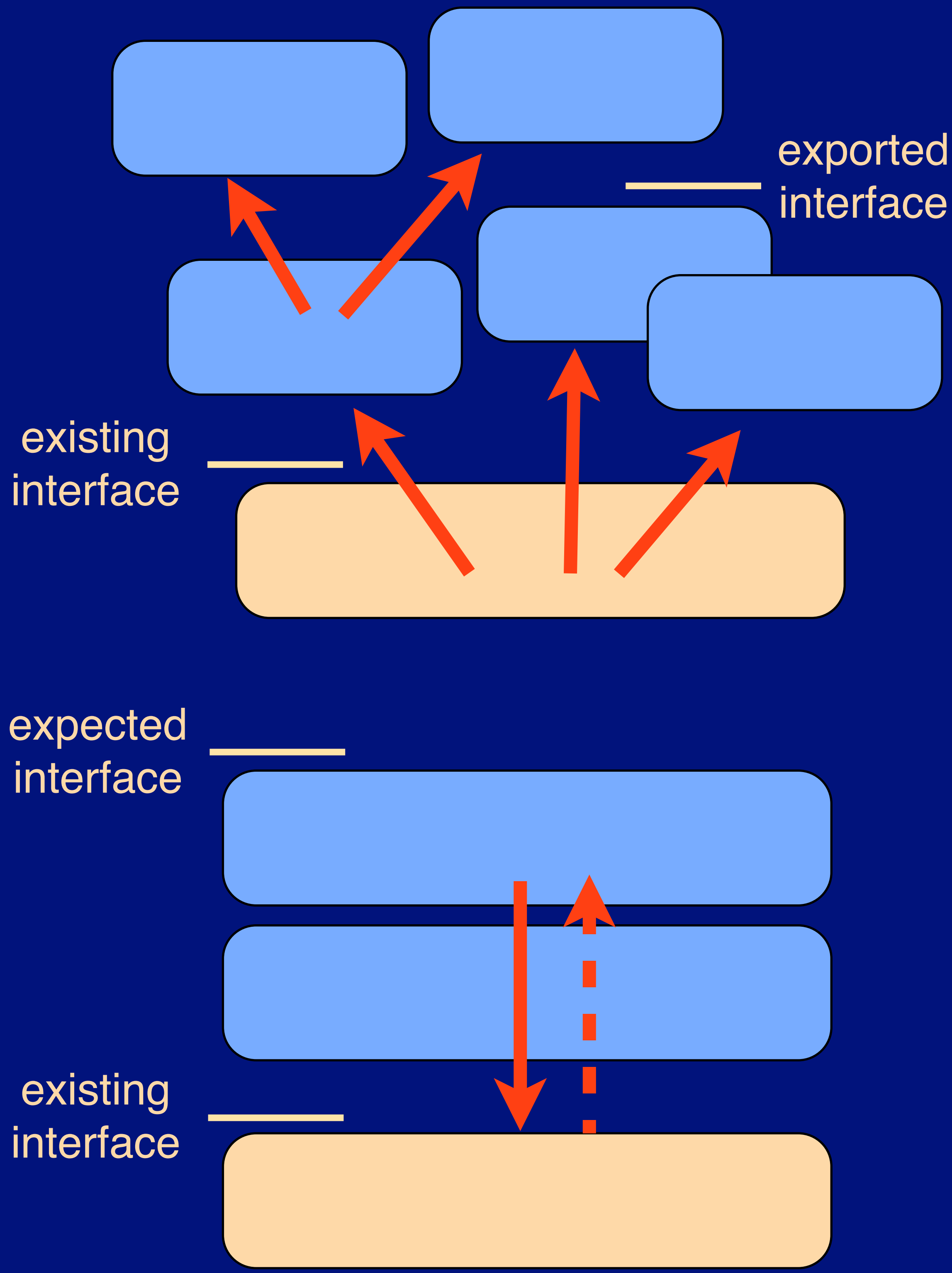


The two faces of virtualization

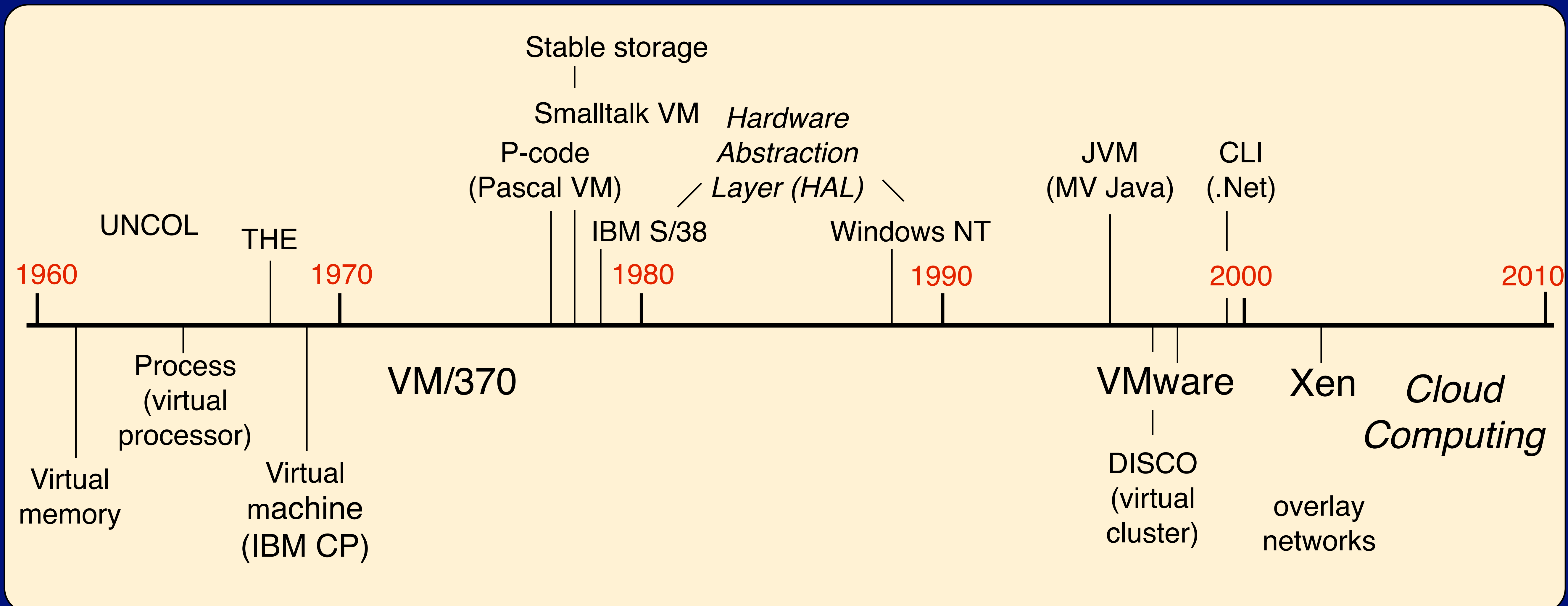
- **Ascending (abstraction)**
  - A tool for resource sharing
  - Creating “high level” resources
  - Multiplexing “low level” resources

Visible interface, hidden implementation
Transforming interfaces (possibly)
Preserving invariants

- **Descending (refinement)**
  - A design tool
  - From specification to implementation
  - Hierarchy of abstract machines

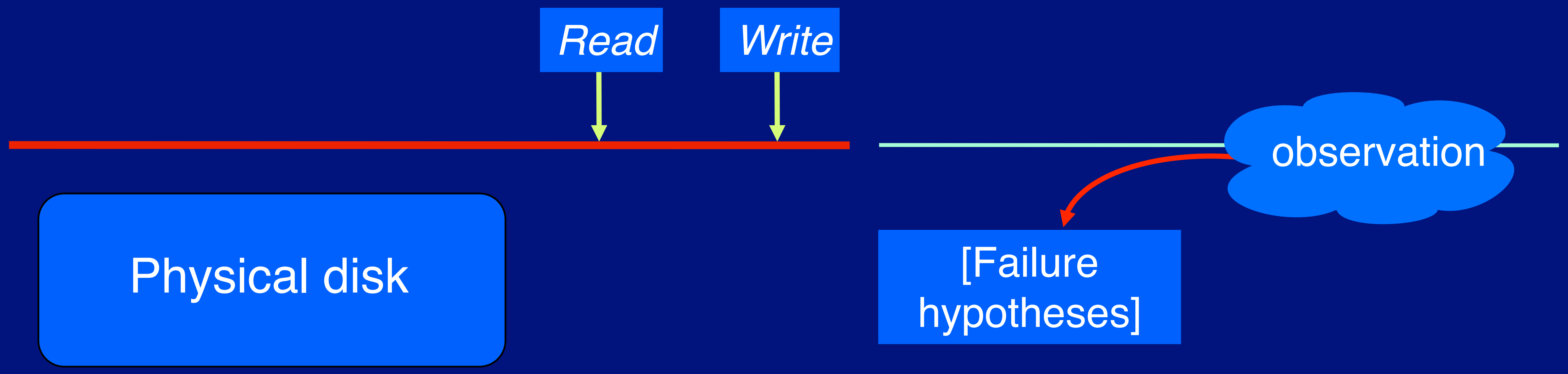


A brief history of virtualization



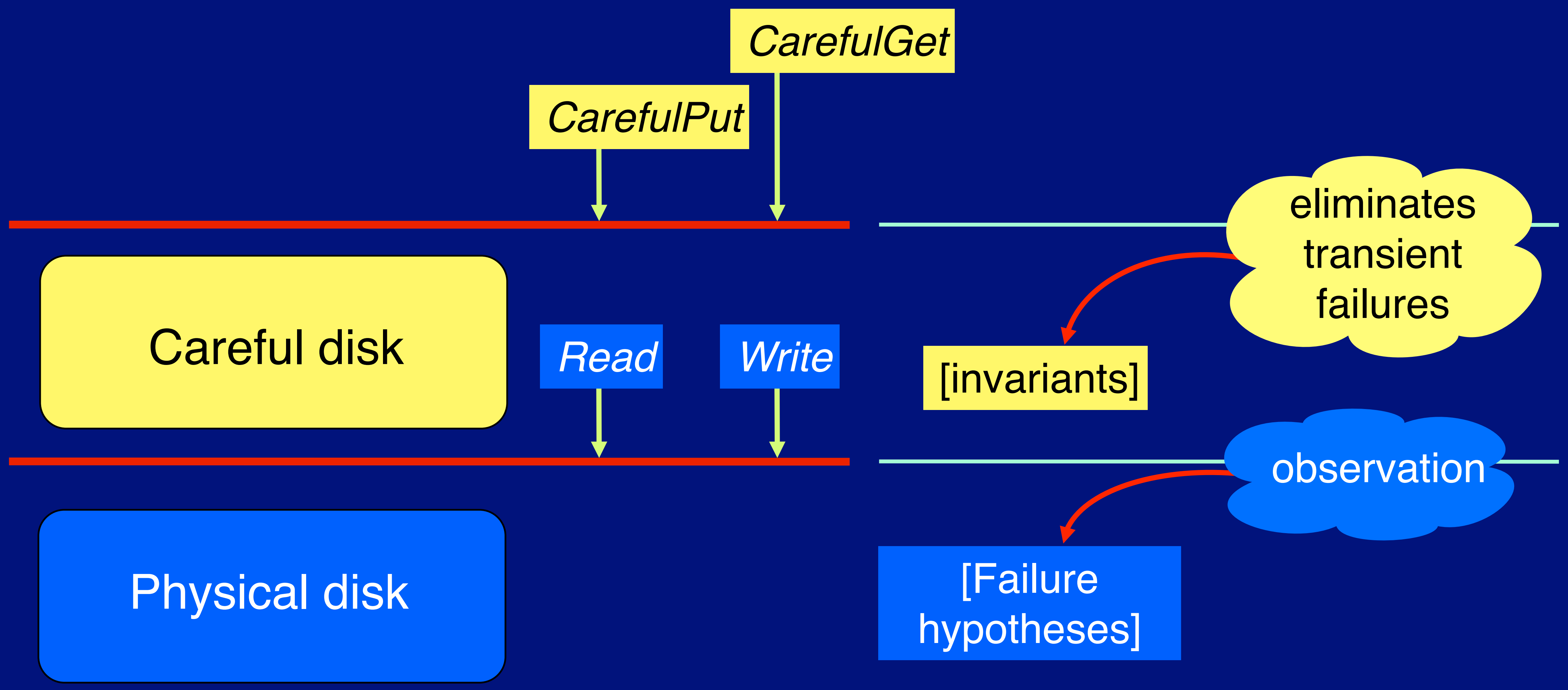
- What can be virtualized?
  - a resource
  - a machine
  - a network
  - an execution environment
  - …

Virtualizing a resource: stable storage



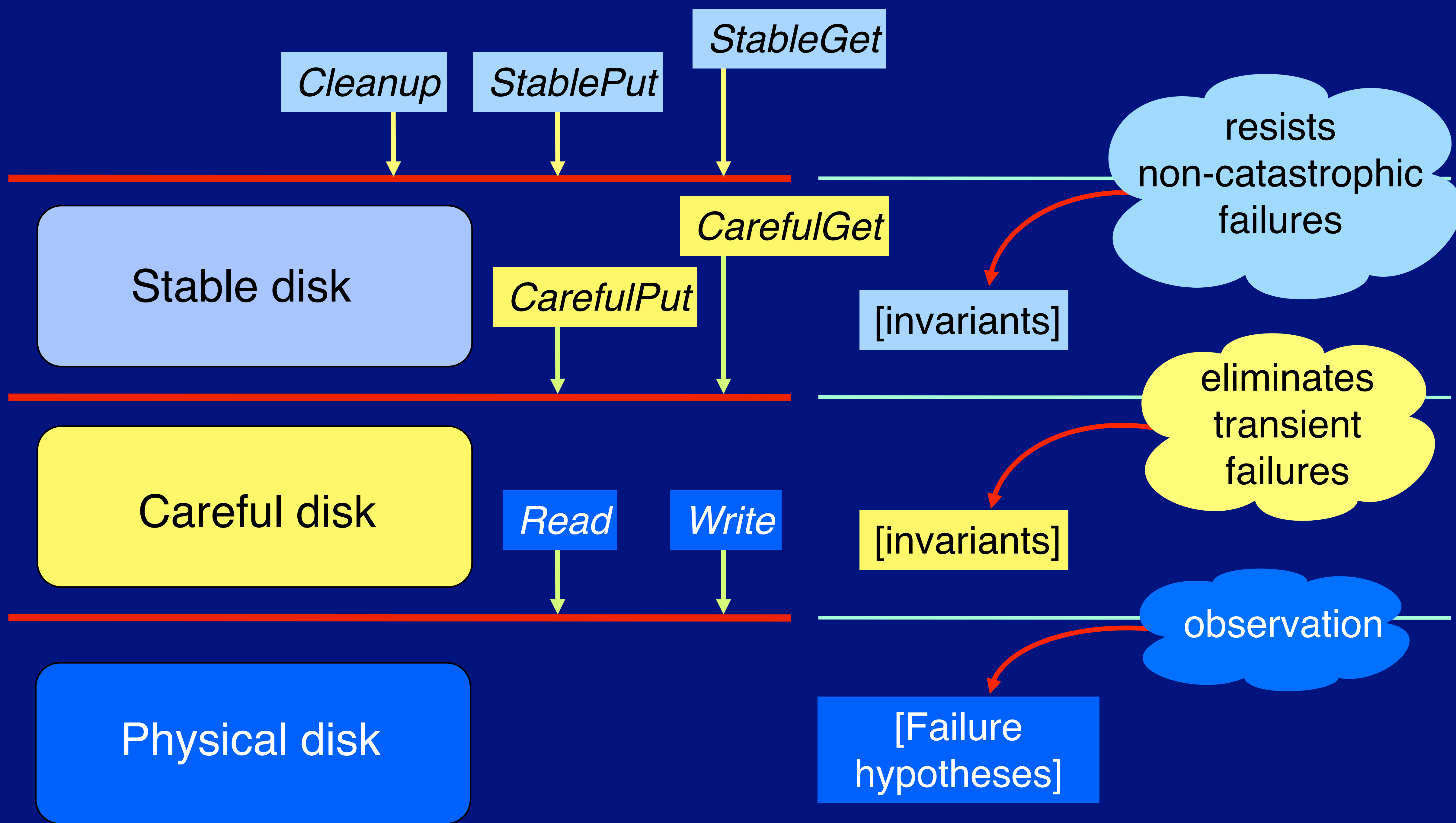
B. W. Lampson and H. E. Sturgis. Crash Recovery in a Distributed Data Storage System, unpublished technical report, Xerox PARC, June 1979, 25 pp.

Virtualizing a resource: stable storage



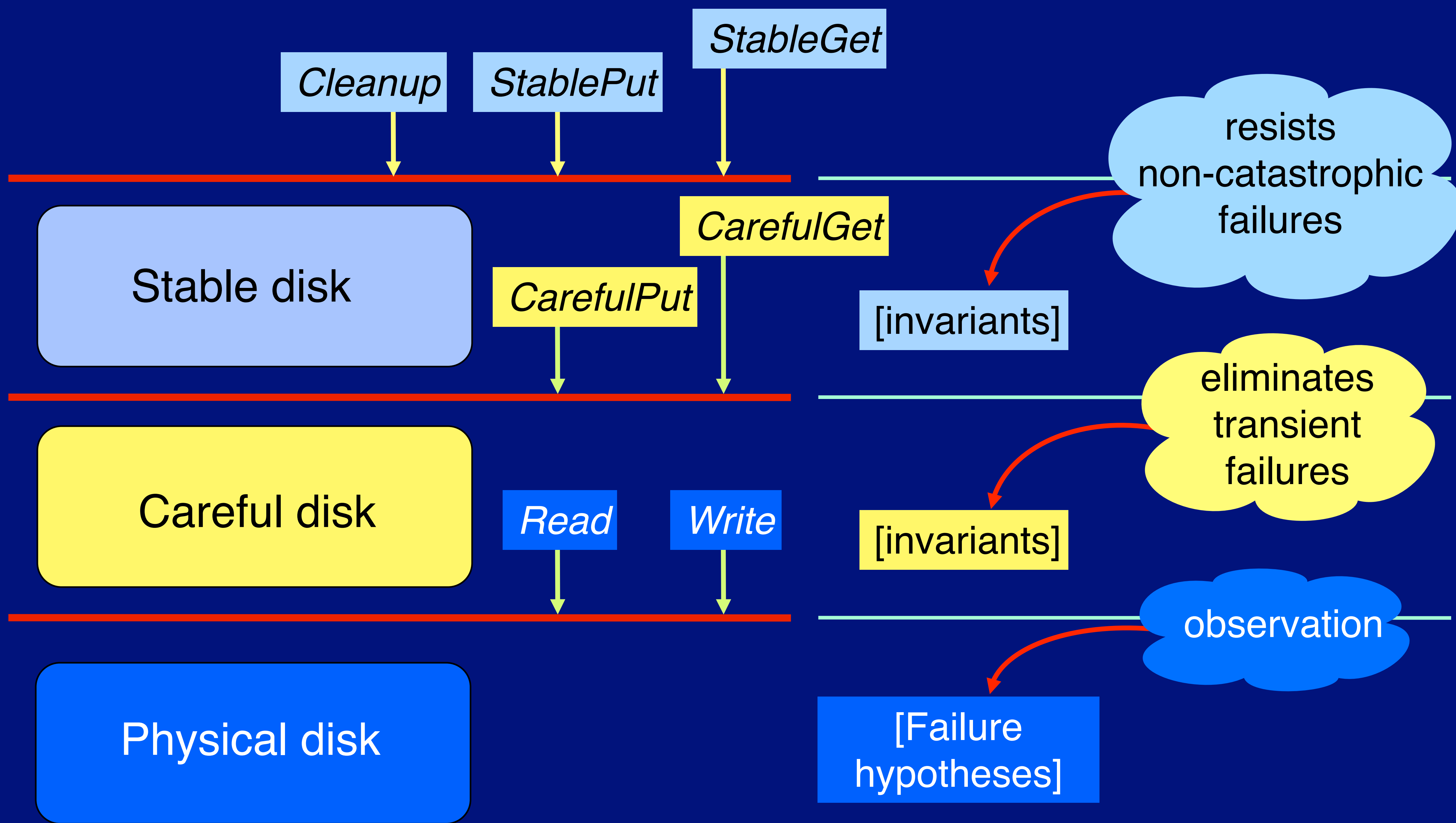
B. W. Lampson and H. E. Sturgis. Crash Recovery in a Distributed Data Storage System, unpublished technical report, Xerox PARC, June 1979, 25 pp.

Virtualizing a resource: stable storage



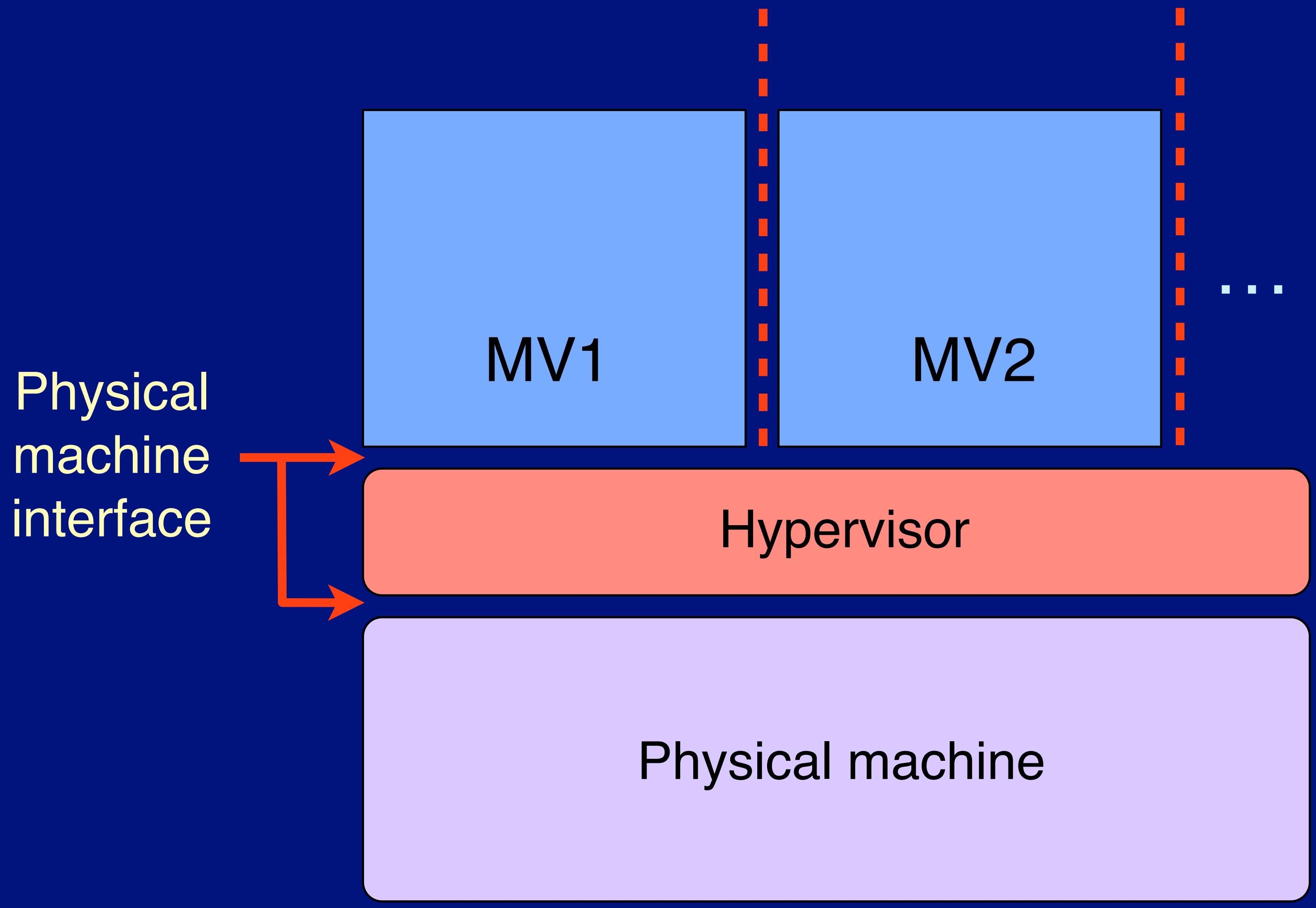
B. W. Lampson and H. E. Sturgis. Crash Recovery in a Distributed Data Storage System, unpublished technical report, Xerox PARC, June 1979, 25 pp.

Virtualizing a resource: stable storage

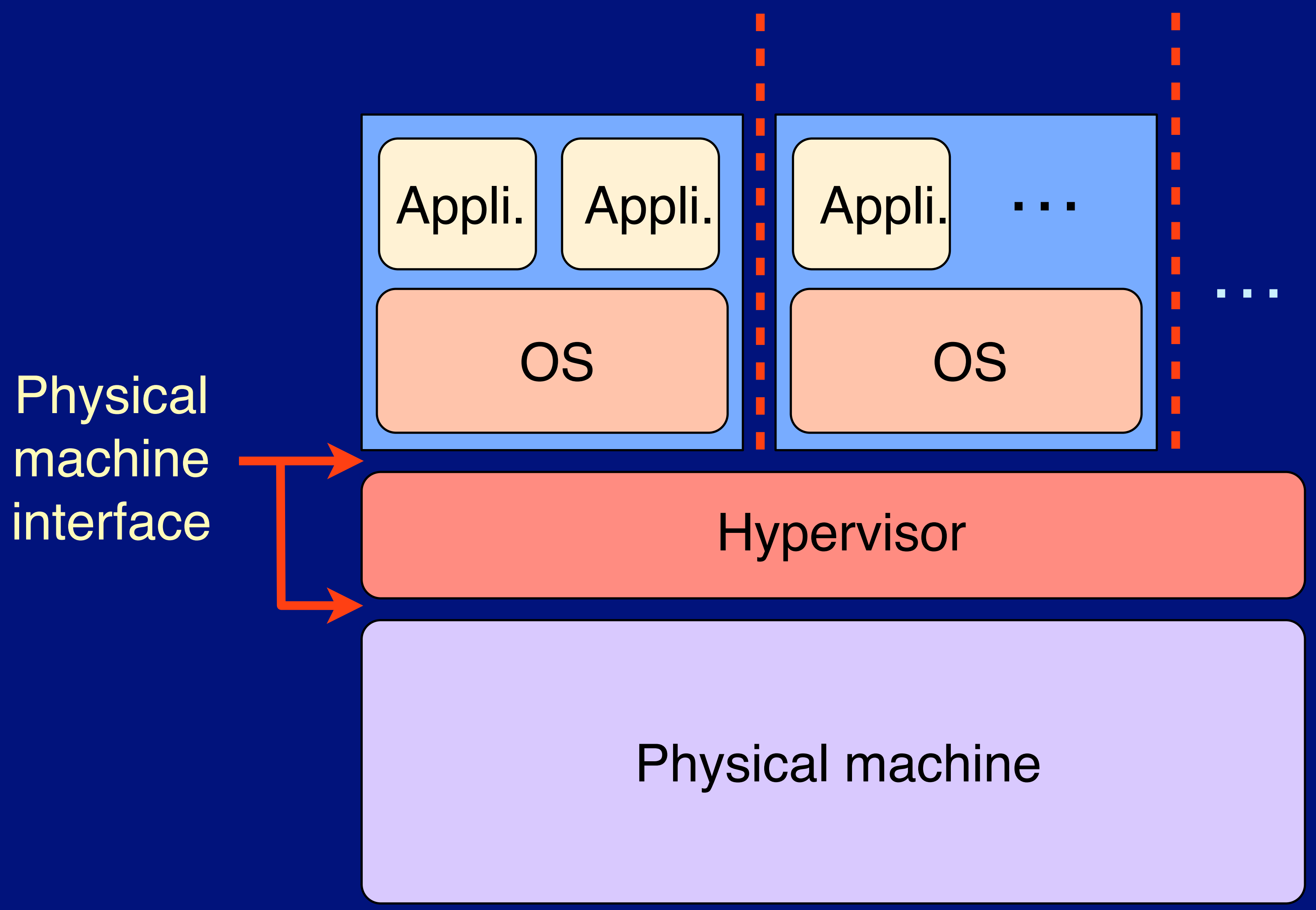


B. W. Lampson, “Atomic Transactions”, in *Distributed Systems—Architecture and Implementation*, ed. Lampson, Paul, and Siegert, LNCS 105, Springer, 1981, pp. 246-265.

“Classic” virtual machines

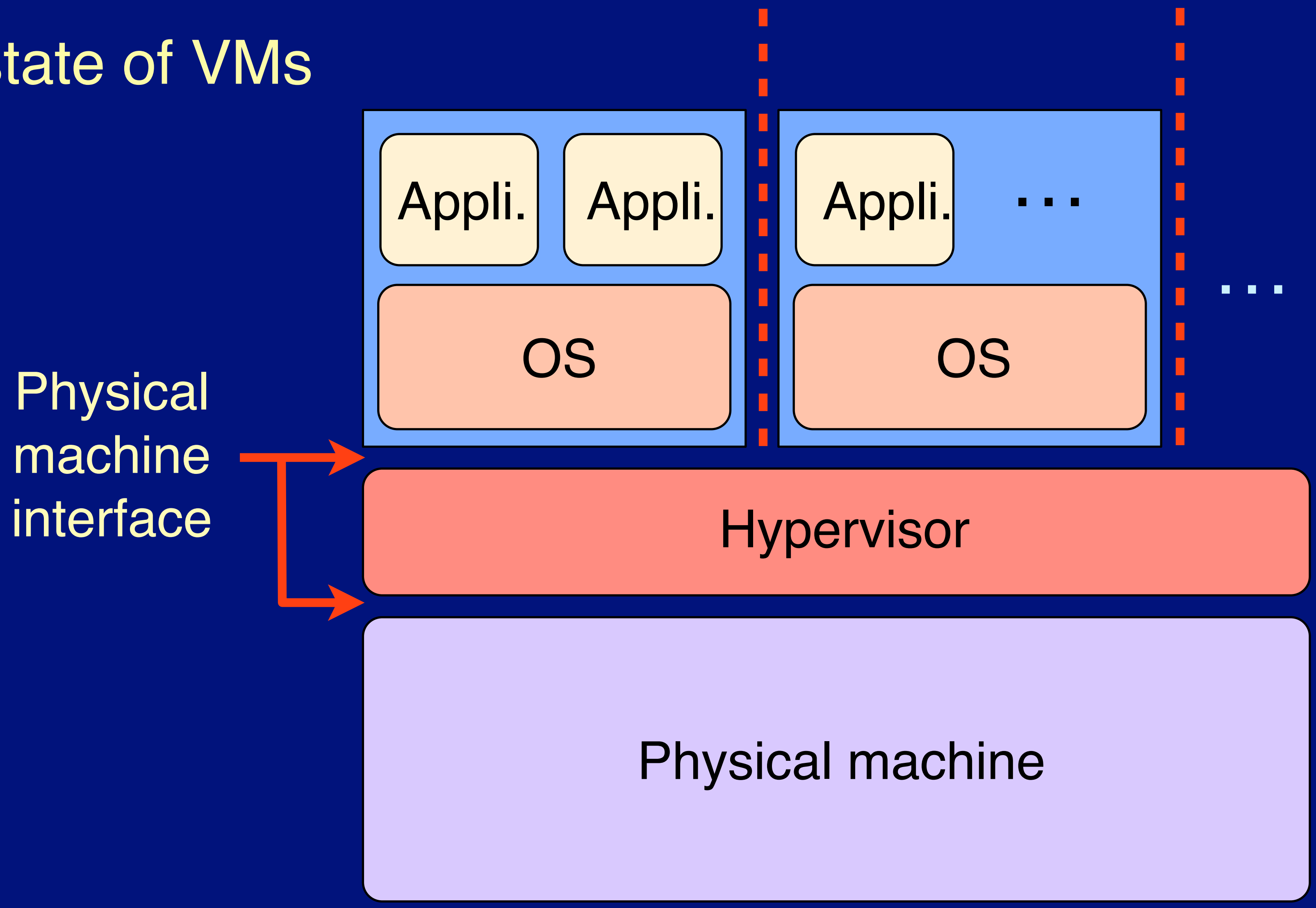


“Classic” virtual machines



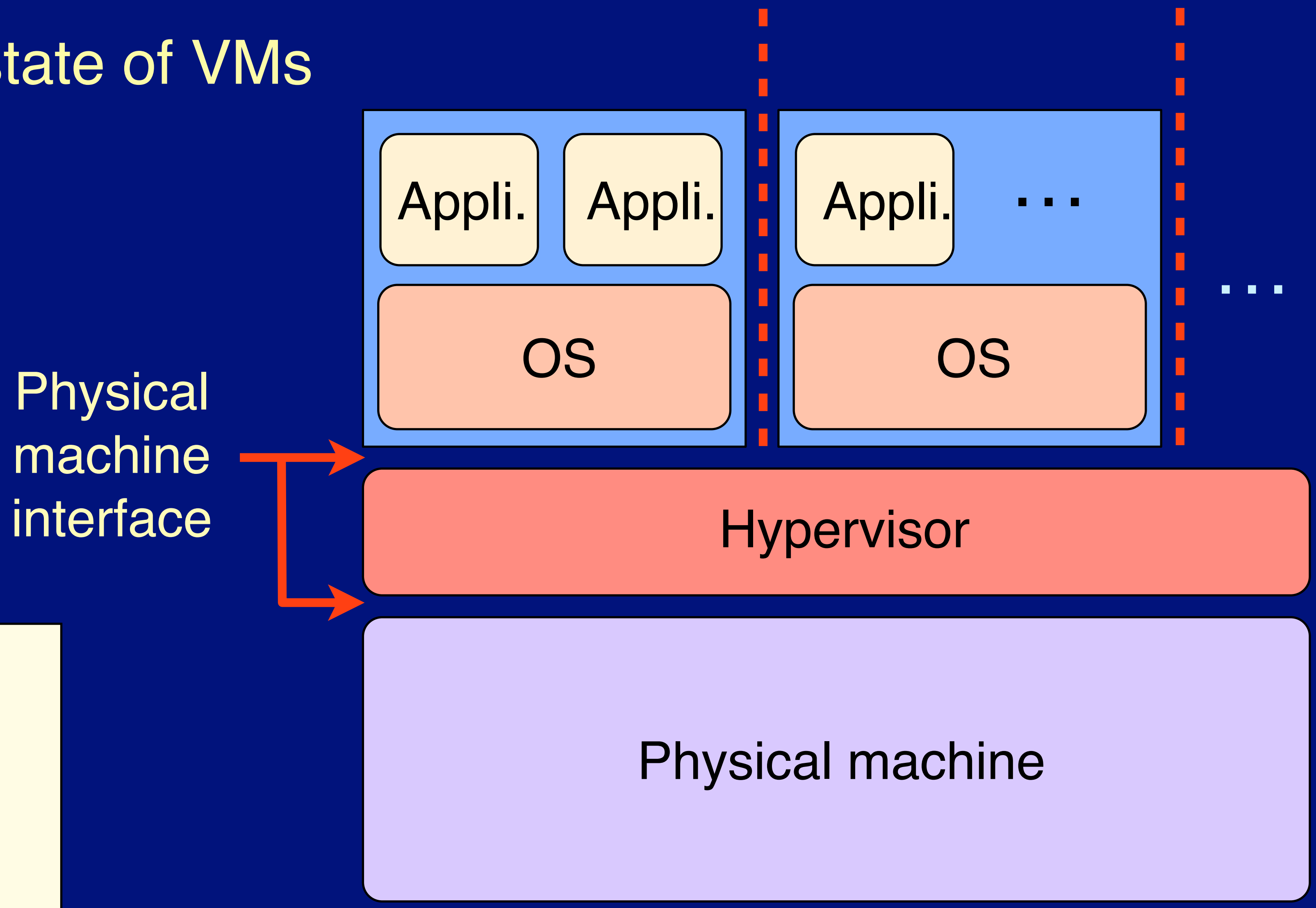
“Classic” virtual machines

- The hypervisor, a super-OS
  - Presents a uniform interface to virtual machines (VMs)
  - Manages (and protects) physical resources
  - Encapsulates internal state of VMs
  - A critical component
    - Isolation
    - Failures
    - Security



“Classic” virtual machines

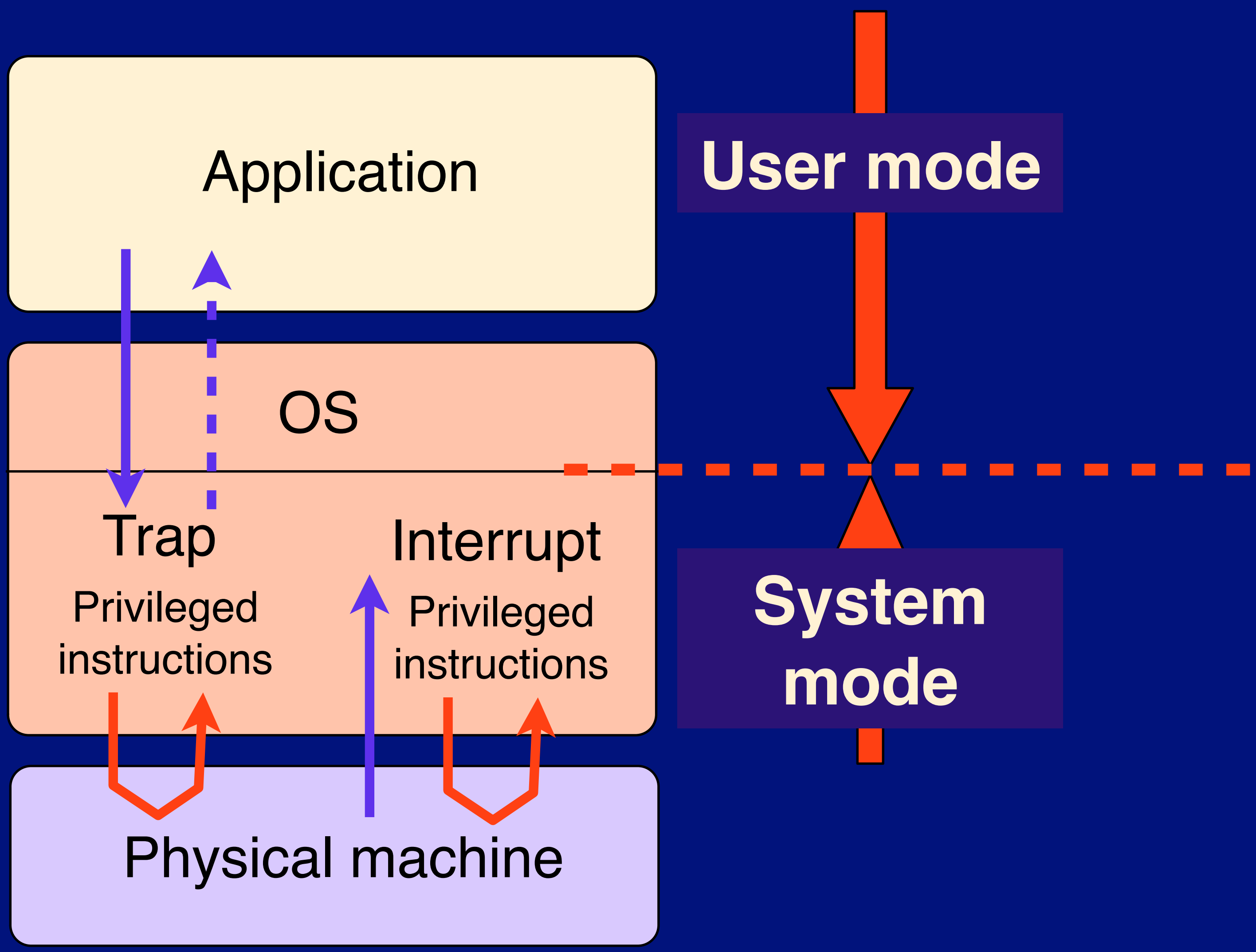
- The hypervisor, a super-OS
  - Presents a uniform interface to virtual machines (VMs)
  - Manages (and protects) physical resources
  - Encapsulates internal state of VMs
  - A critical component
    - Isolation
    - Failures
    - Security



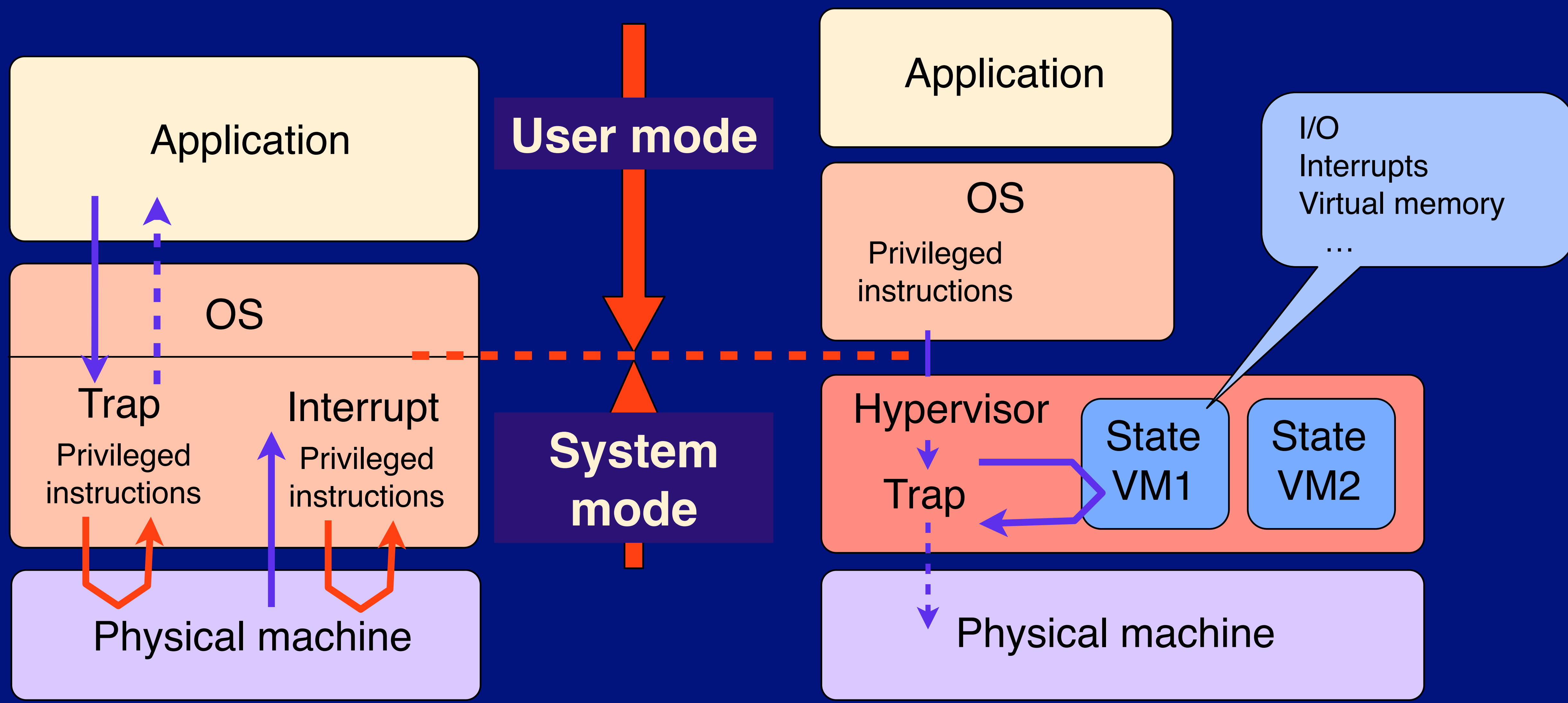
James E. Smith, Ravi Nair, *Virtual Machines*, Morgan Kaufmann, 2005

Virtualization Technologies, *IEEE Computer*, May 2005

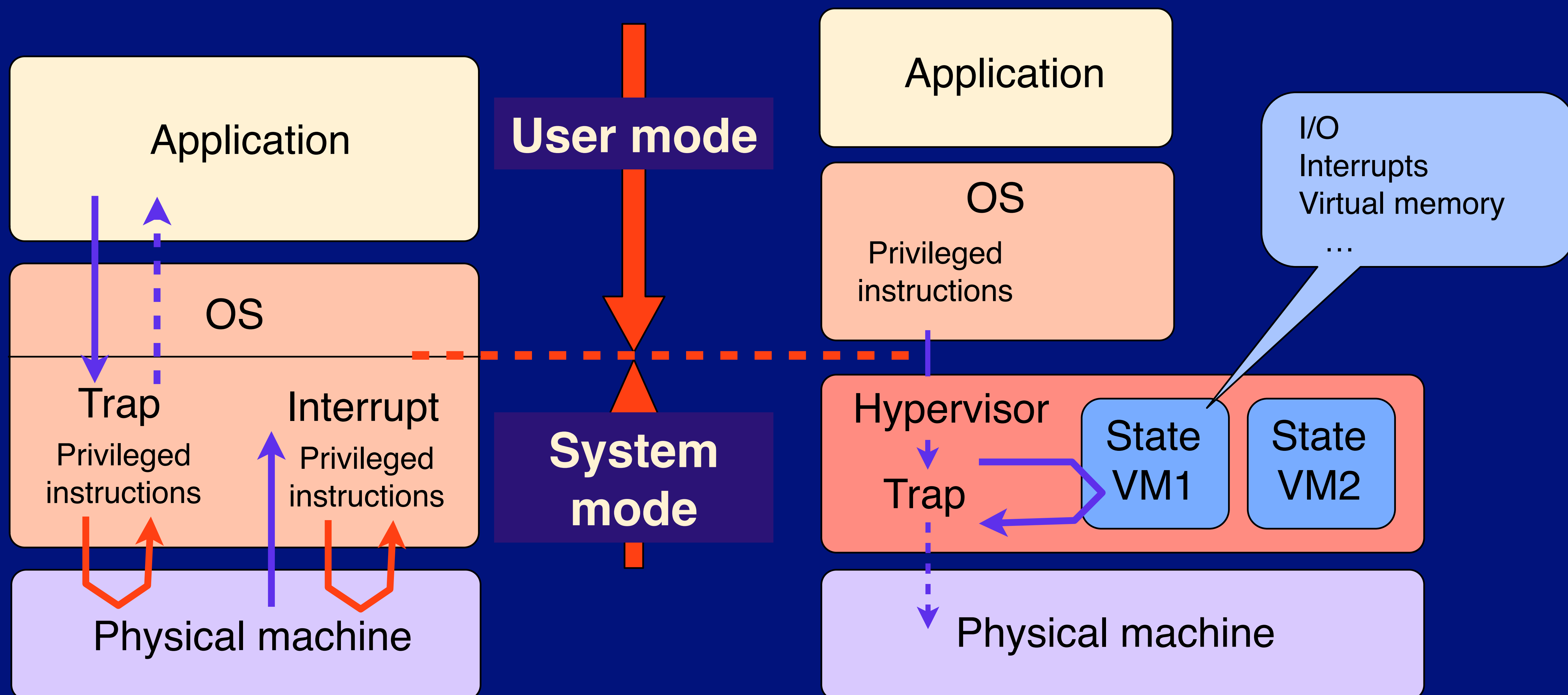
Virtual machines : how it works



Virtual machines : how it works



Virtual machines : how it works



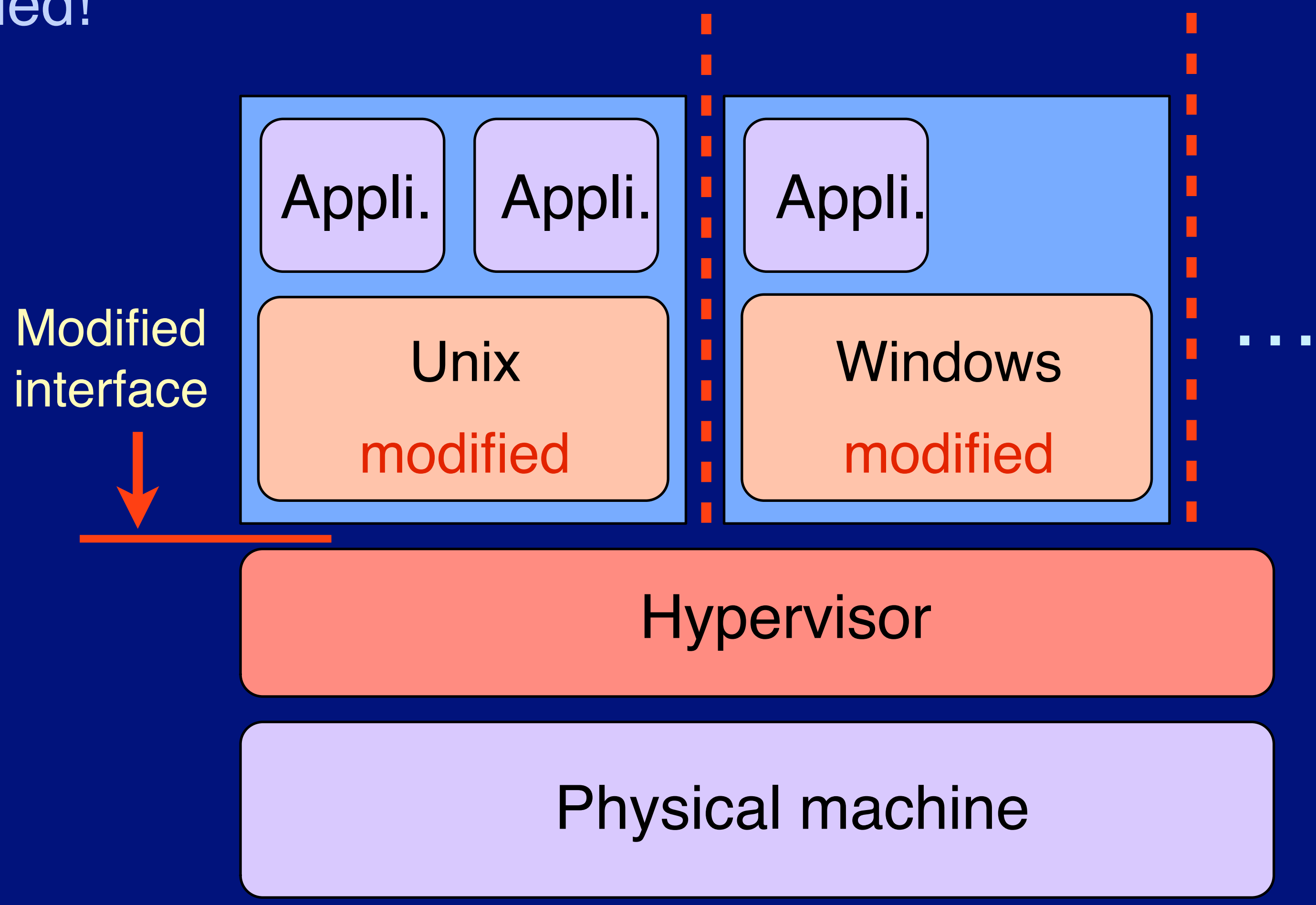
- A problem …

  On current machines (IA-32, etc.), the effect of some instructions *depends on the current mode* (system or user)

  An ISA containing such instructions *cannot* be virtualized!

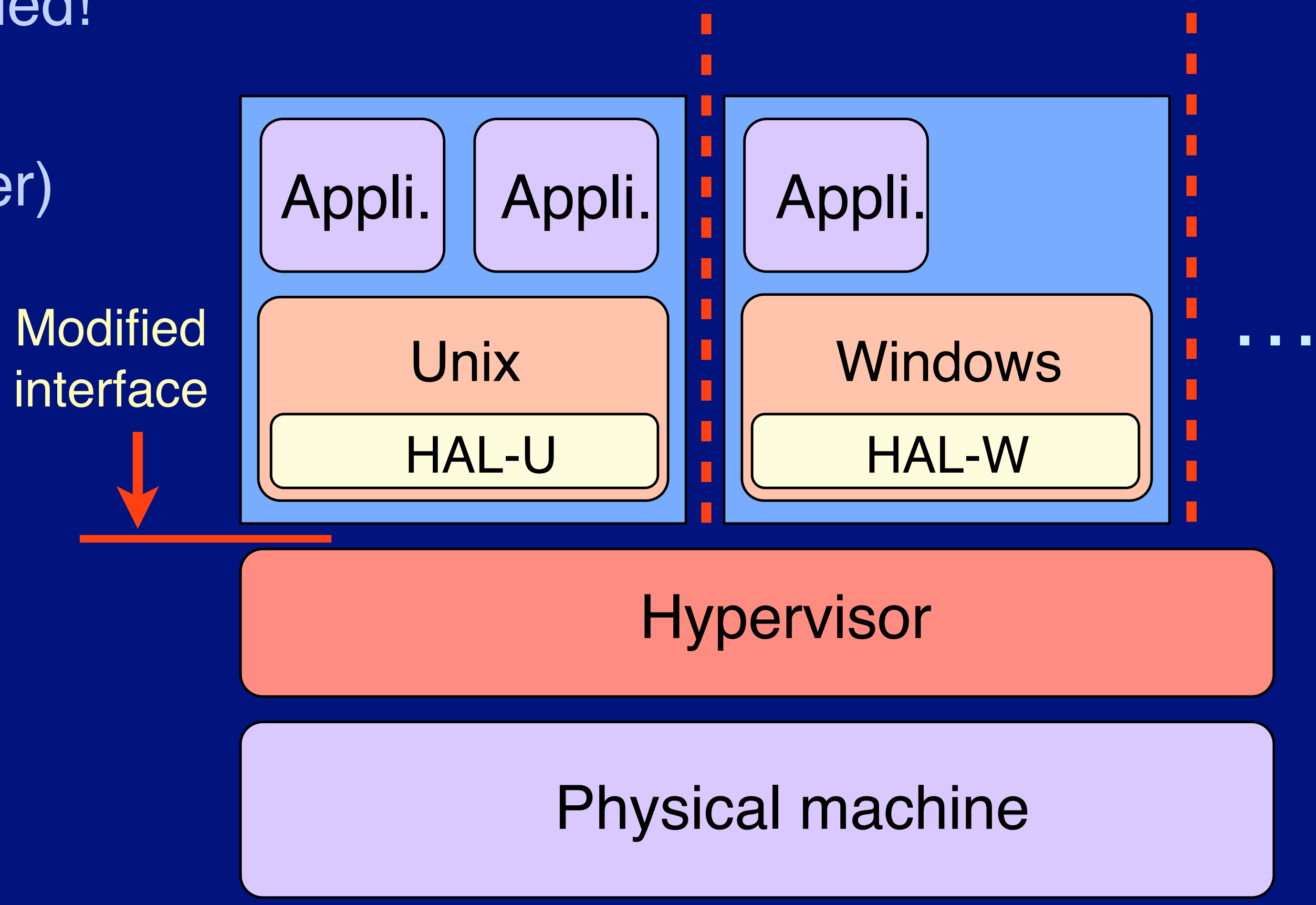
Virtual machines

- How to escape the problem of multi-mode instructions?
  - Paravirtualization: changing the hypervisor's interface
    - Replace non-virtualizable instructions
    - The interface is no longer that of the physical machine
    - Therefore OSs must be modified!



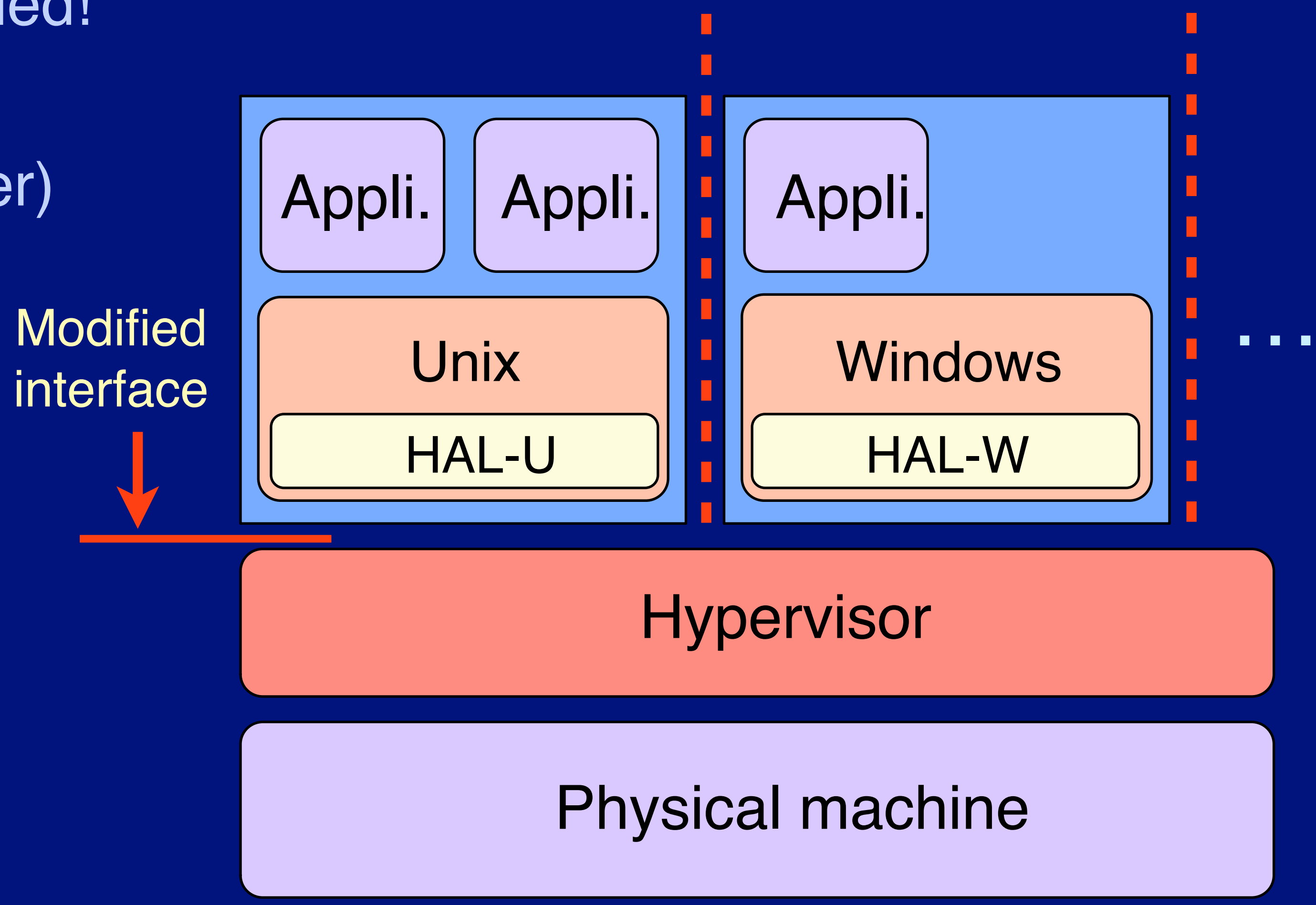
Virtual machines

- How to escape the problem of multi-mode instructions?
  - Paravirtualization: changing the hypervisor's interface
    - Replace non-virtualizable instructions
    - The interface is no longer that of the physical machine
    - Therefore OSs must be modified!
    - In practice, only the HAL
      (Hardware Abstraction Layer)
      needs to be modified



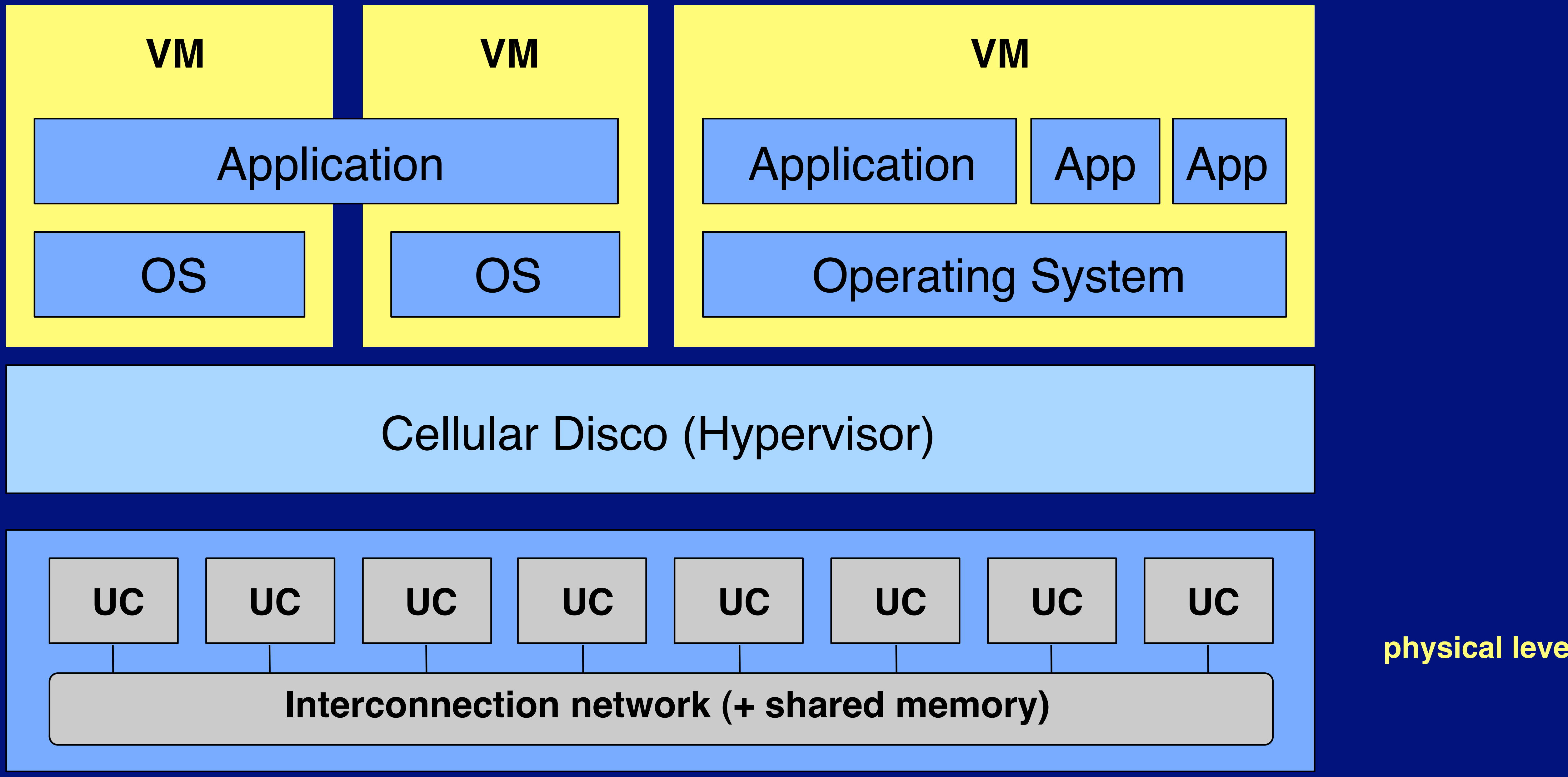
Virtual machines

- How to escape the problem of multi-mode instructions?
  - Paravirtualization: changing the hypervisor's interface
    - Replace non-virtualizable instructions
    - The interface is no longer that of the physical machine
    - Therefore OSs must be modified!
    - In practice, only the HAL
      (Hardware Abstraction Layer)
      needs to be modified

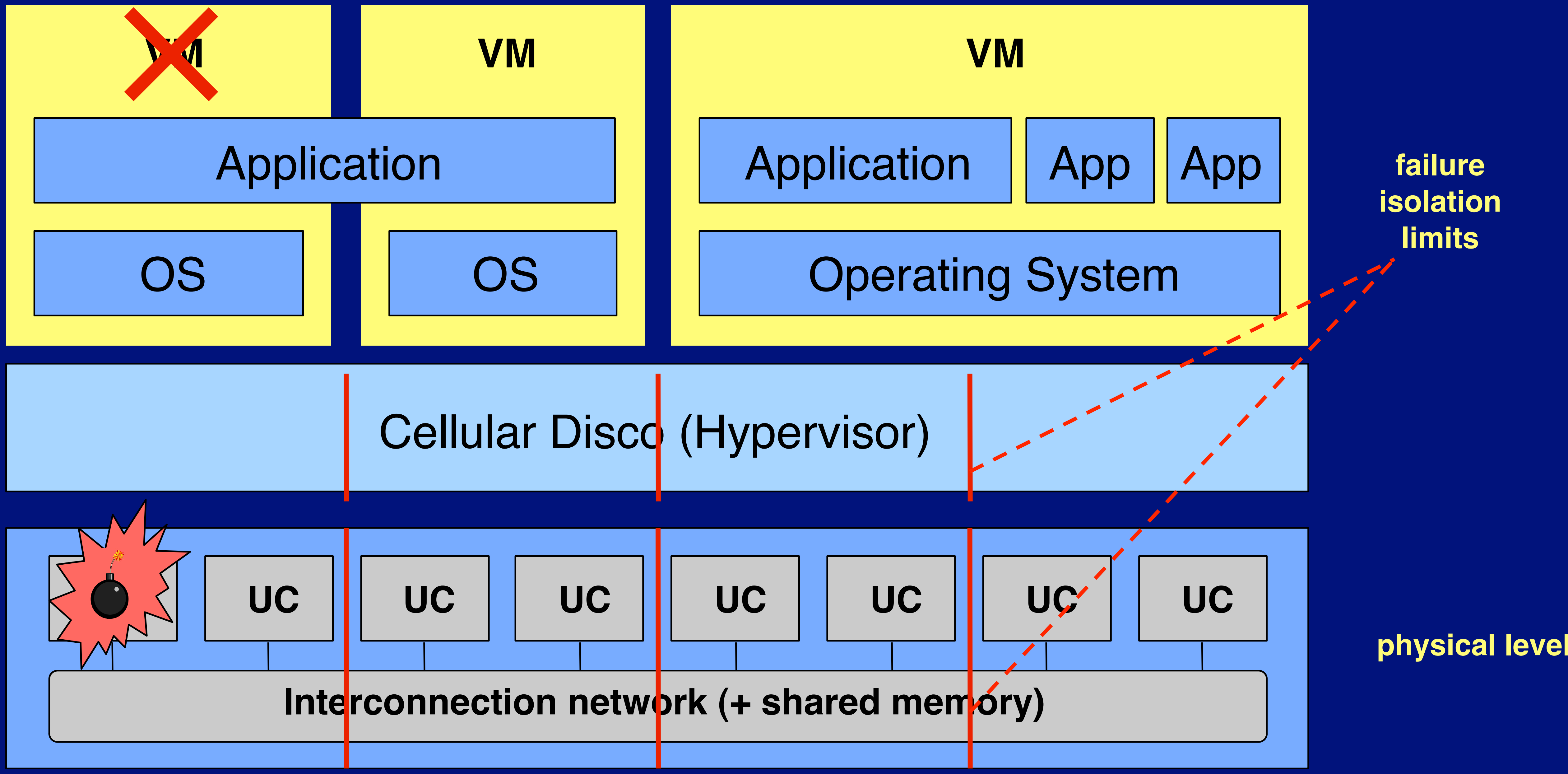


  - Dynamic translation of binary code:
    - Replacing non-virtualizable instructions in real time
  - In the future:
    - New processors will be designed for virtualization

Virtual clusters



Virtual clusters



K. Govil, D. Teodosiu, Y. Huang, M. Rosenblum. Cellular Disco: Resource Management Using Virtual Clusters on Shared-Memory Multiprocessors, *ACM Trans. on Computer Systems*, 18(3), Aug. 2000

Cloud computing

- An old vision …

  "… *computing may someday be organized as a public utility just as the telephone system is a public utility*"

  John McCarthy, 1961

- … close to being achieved?
- Virtualizing on a large scale

  hardware: *Infrastructure as a Service* (Amazon EC2)

  execution environment: *Platform as a Service* (Microsoft Azure)

  application support: *Software as a Service* (Google Docs)

- A new economic model …

  … but potential problems

- An open research area

Questions on Clouds

- What is new, anyway?

Questions on Clouds

- What is new, anyway?
  - "Elasticity": the client pays what he uses, fine grain accounting
    - For the client: economy, no risk of over- / under-provisioning, potentially unlimited capacity
    - For the provider: gain (scale effect, statistical multiplexing, amortizing investments)
  - Reactivity to variations of the load

D. Owens, "Securing Elasticity in the Cloud", *Comm. of the ACM*, vol. 53, no 6, June 2010

Questions on Clouds

- What is new, anyway?
  - "Elasticity": the client pays what he uses, fine grain accounting
    - For the client: economy, no risk of over- / under-provisioning, potentially unlimited capacity
    - For the provider: gain (scale effect, statistical multiplexing, amortizing investments)
  - Reactivity to variations of the load

D. Owens, "Securing Elasticity in the Cloud", *Comm. of the ACM*, vol. 53, no 6, June 2010

- What risks and problems?
  - Technical limits: evolution, large scale, latency
  - Loss of control over data (location, security, …)
  - No significant cost reduction without sacrificing
    - performance guarantees
    - availability guarantees
    - security guarantees

D. Durkee, "Why Cloud Computing Will Never Be Free", *Comm. of the ACM*, vol. 53, no 5, May 2010

Virtualization in embedded systems

- Specific constraints
  - Increasingly complex applications
  - Need to:
    - Control performance
    - Reduce the size of the trusted base

Virtualization in embedded systems

- Specific constraints
  - Increasingly complex applications
  - Need to:
    - Control performance
    - Reduce the size of the trusted base
- A new life for microkernels
  - The microkernel as low-level hypervisor
  - Customized operating systems for specific applications
    - "Virtual devices" (a device + its customized OS)
  - Device drivers need not be in the trusted base
  - Critical components may be isolated in VMs

G. Heiser, "The Role of Virtualization in Embedded Systems", *Proc. First Workshop on Isolation and Integration in Embedded Systems (IIES'08)*, pp 11-16, April 2008

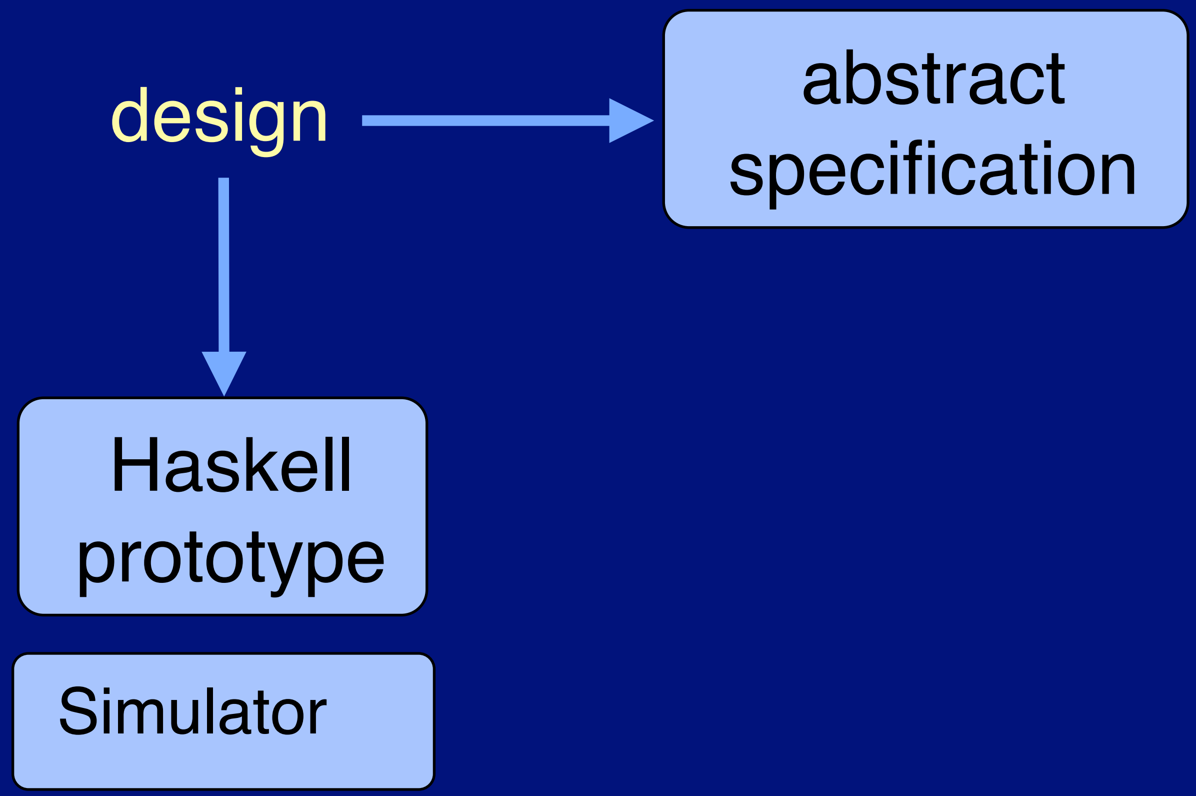
A trusted microkernel

- A version of the L4 microkernel
  - A virtual machine providing a simplified image of the hardware
- Difficulties
  - Asynchronism
  - Memory management (pointers)
  - Direct access to hardware functions (MMU, …)

Gerwin Klein et al., "seL4: Formal Verification of an Operating-System Kernel", *Communications of the ACM*, vol. 53, no 6, pp. 107-115, June 2010

A trusted microkernel

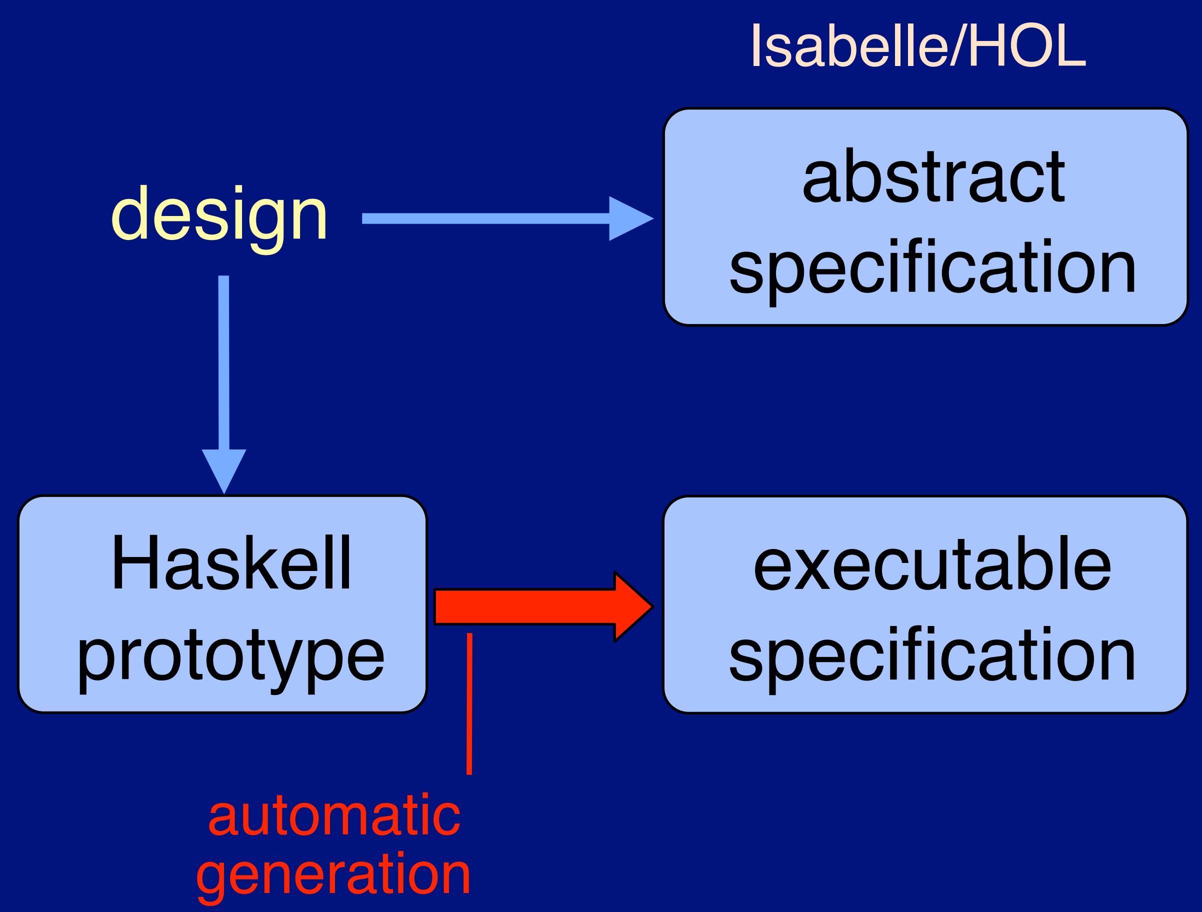
- A version of the L4 microkernel
  - A virtual machine providing a simplified image of the hardware
- Difficulties
  - Asynchronism
  - Memory management (pointers)
  - Direct access to hardware functions (MMU, …)



Gerwin Klein et al., “seL4: Formal Verification of an Operating-System Kernel”, *Communications of the ACM*, vol. 53, no 6, pp. 107-115, June 2010

A trusted microkernel

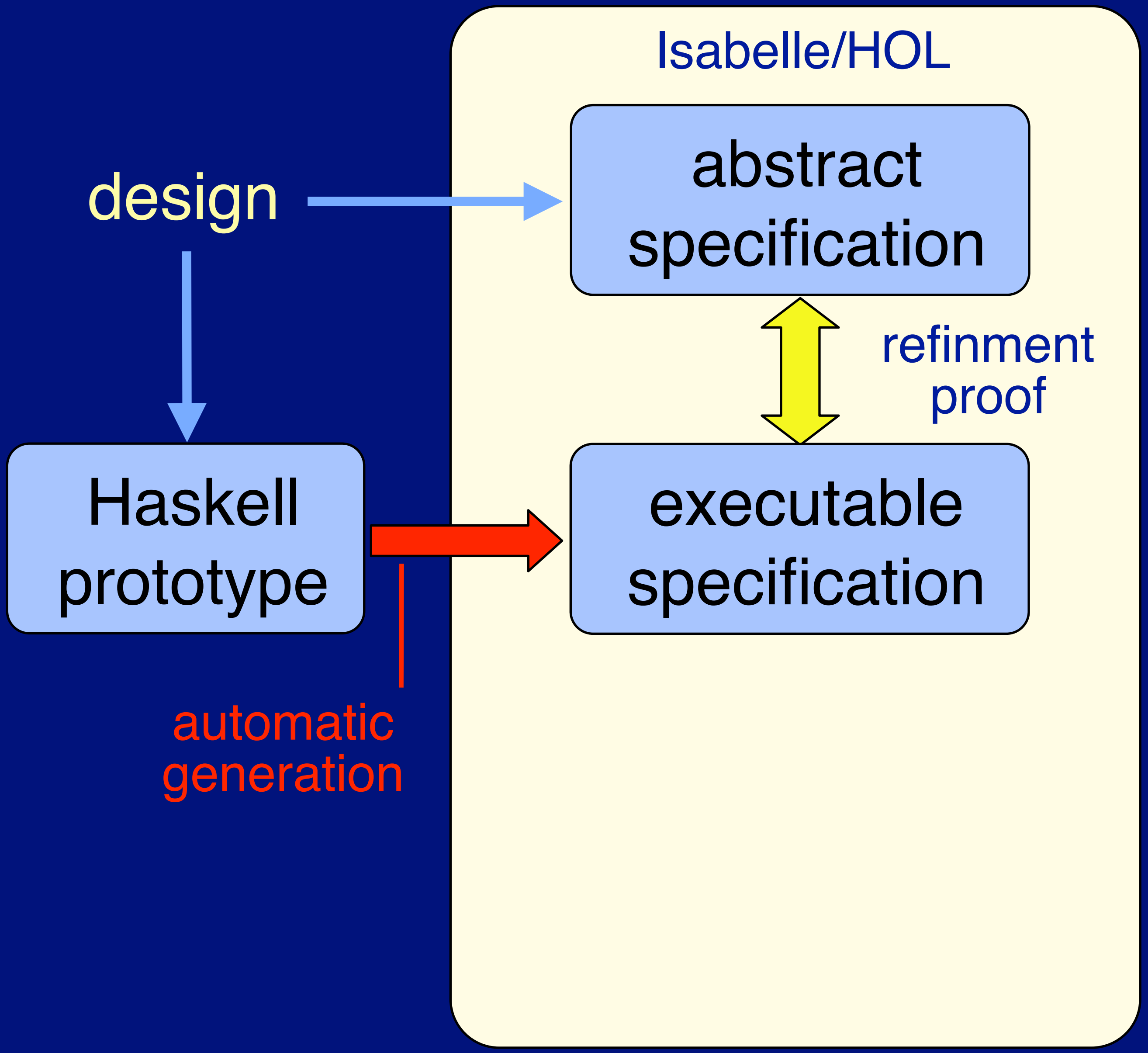
- A version of the L4 microkernel
  - A virtual machine providing a simplified image of the hardware
- Difficulties
  - Asynchronism
  - Memory management (pointers)
  - Direct access to hardware functions (MMU, …)



Gerwin Klein et al., "seL4: Formal Verification of an Operating-System Kernel", *Communications of the ACM*, vol. 53, no 6, pp. 107-115, June 2010

A trusted microkernel

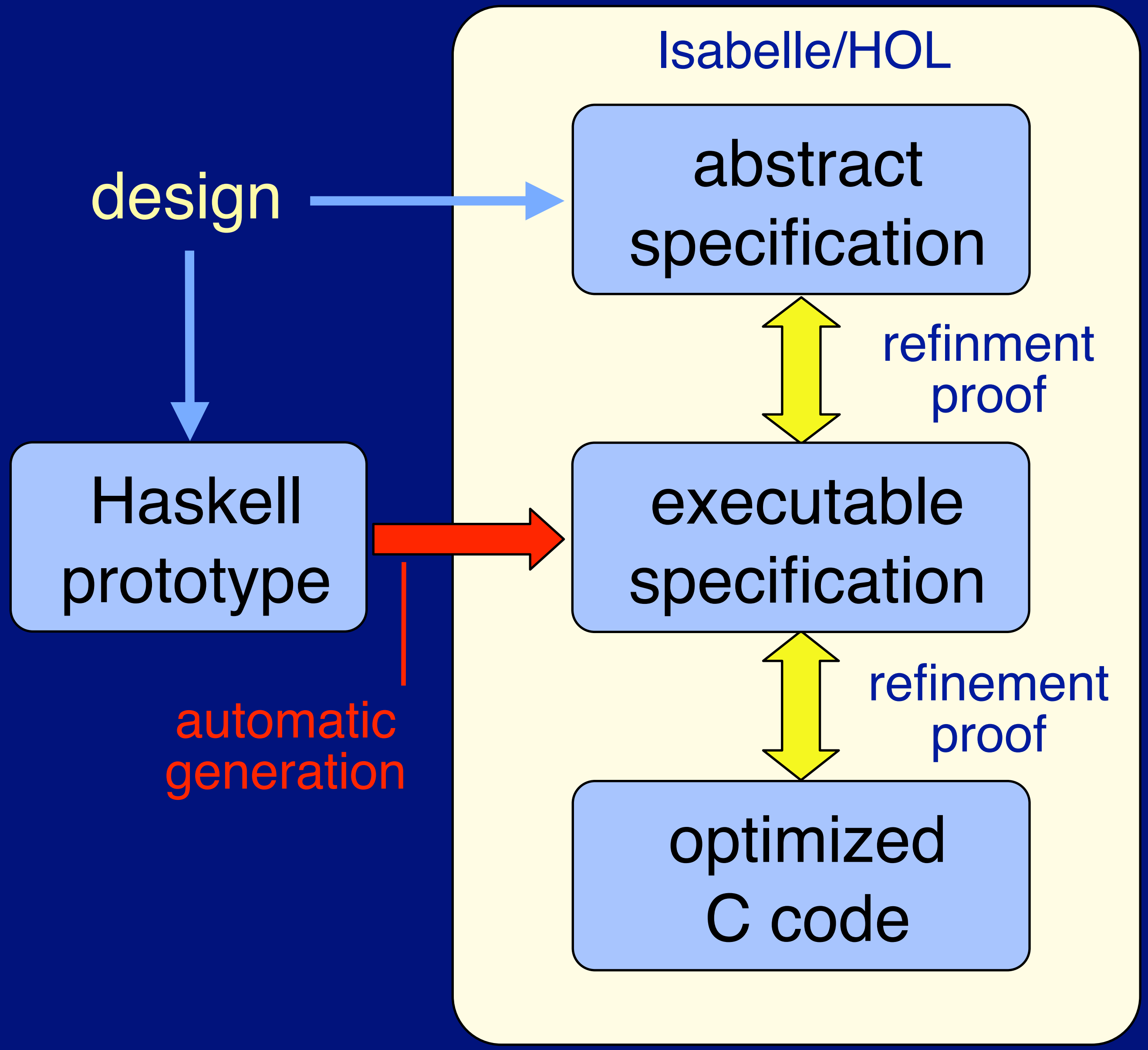
- A version of the L4 microkernel
  - A virtual machine providing a simplified image of the hardware
- Difficulties
  - Asynchronism
  - Memory management (pointers)
  - Direct access to hardware functions (MMU, …)



Gerwin Klein et al., "seL4: Formal Verification of an Operating-System Kernel", *Communications of the ACM*, vol. 53, no 6, pp. 107-115, June 2010

A trusted microkernel

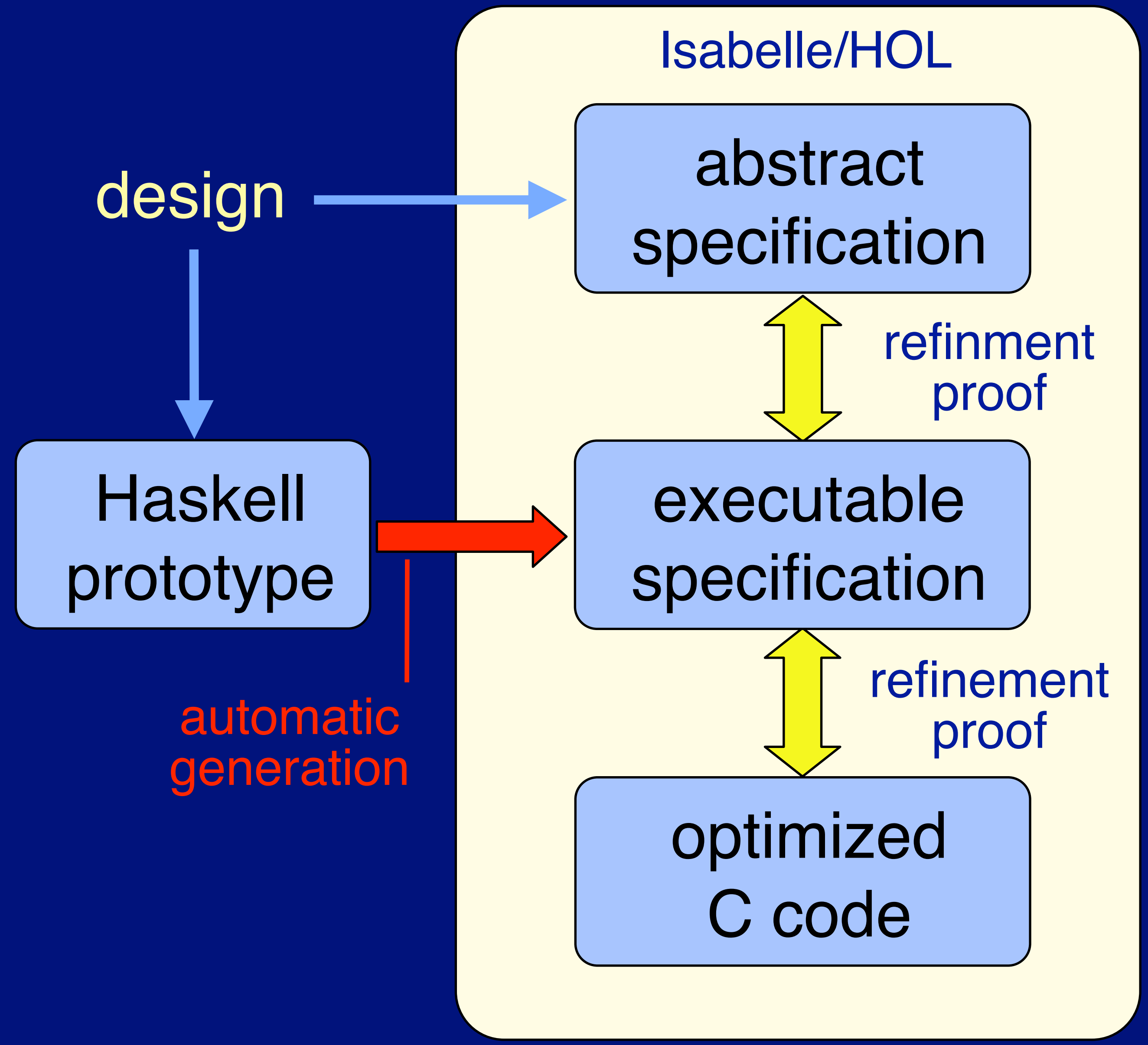
- A version of the L4 microkernel
  - A virtual machine providing a simplified image of the hardware
- Difficulties
  - Asynchronism
  - Memory management (pointers)
  - Direct access to hardware functions (MMU, …)



Gerwin Klein et al., "seL4: Formal Verification of an Operating-System Kernel", *Communications of the ACM*, vol. 53, no 6, pp. 107-115, June 2010

A trusted microkernel

- A version of the L4 microkernel
  - A virtual machine providing a simplified image of the hardware
- Difficulties
  - Asynchronism
  - Memory management (pointers)
  - Direct access to hardware functions (MMU, …)
- seL4, base for a "microvisor"
  - OKL4 (Open Kernel Labs)
  - Installed base: one billion …



Gerwin Klein et al., "seL4: Formal Verification of an Operating-System Kernel", *Communications of the ACM*, vol. 53, no 6, pp. 107-115, June 2010

Advances and challenges for virtualization

- Virtualization born again and extended
  - From clouds to embedded systems
  - “De-materialized” platforms and applications
    - Portable across supports and locations
  - Tools for global resource management
  - An environment for experiments

Advances and challenges for virtualization

- Virtualization born again and extended
  - From clouds to embedded systems
  - "De-materialized" platforms and applications
    - Portable across supports and locations
  - Tools for global resource management
  - An environment for experiments
- Challenges
  - For the user
    - Control over management and data
    - Guarantees of availability and security
  - For the designer
    - Modeling and verification of hypervisors
    - Autonomous administration of large infrastructures
    - Managing multiple virtual environments

Composition (and decomposition)

- A deceptively simple objective …
  - Composing a system from elementary pieces
  - Reusable and interchangeable elements (“standard replacement”)
  - Visible interface, hidden implementation

M. D. McIlroy (1968) “Mass Produced Software Components”, *in* P. Naur and B. Randell, eds., *Software Engineering*, NATO Science Committee

Composition (and decomposition)

- A deceptively simple objective …
  - Composing a system from elementary pieces
  - Reusable and interchangeable elements (“standard replacement”)
  - Visible interface, hidden implementation
- … but a road fraught with pitfalls
  - Conceptual
    - Model(s)
    - Expressing global description
    - Guarantees
  - Practical
    - Configuration and deployment
    - Evolution management
    - Infrastructures

> M. D. McIlroy (1968) “Mass Produced Software Components”, *in* P. Naur and B. Randell, eds., *Software Engineering*, NATO Science Committee

Properties of composition

- Composability

  The properties of each component are preserved in the compound system

Properties of composition

- Composability

  The properties of each component are preserved in the compound system

→

Separating interface from implementation

Respecting rules of “correct assembly”

Properties of composition

- **Composability**

  The properties of each component are preserved in the compound system

- **Compositionality**

  The properties of a compound system may be derived from those of the components and from the assembly rules

Separating interface from implementation

Respecting rules of “correct assembly”

Properties of composition

- **Composability**

  The properties of each component are preserved in the compound system

  → Separating interface from implementation

  Respecting rules of “correct assembly”

Much, much harder!

- **Compositionality**

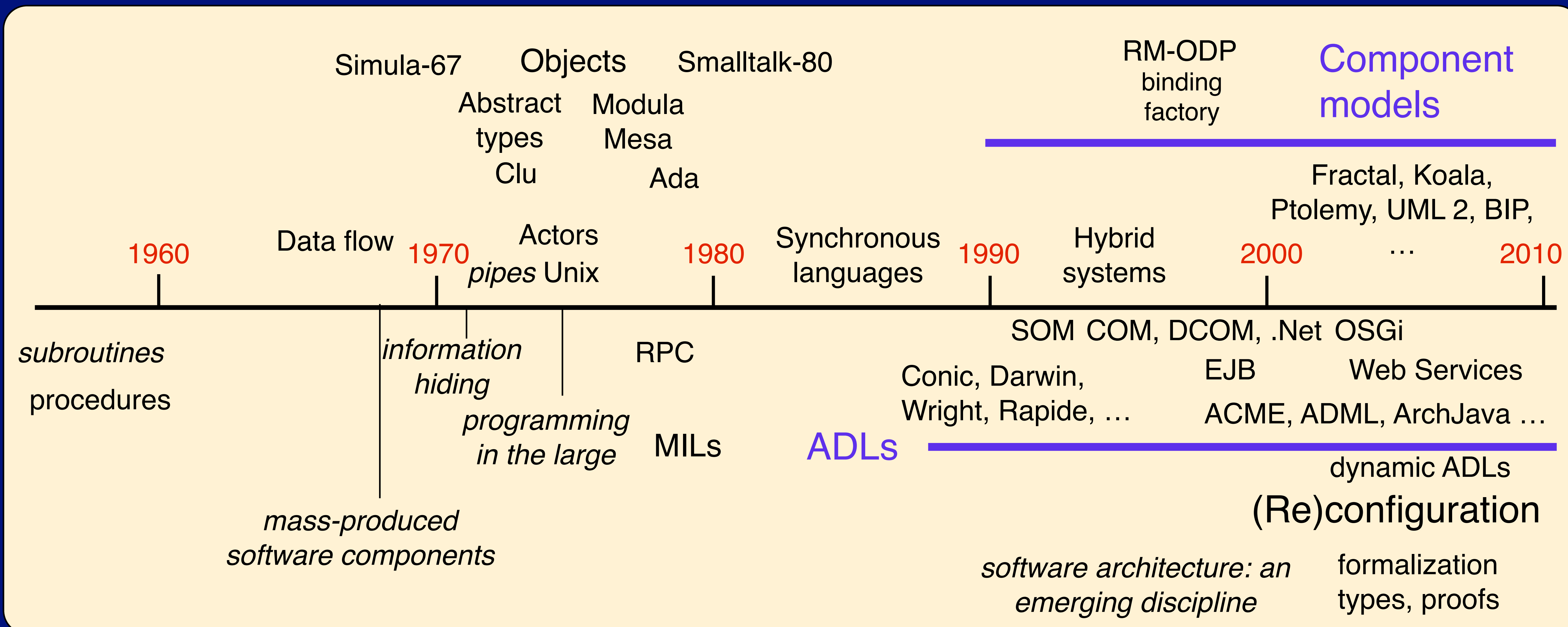
  The properties of a compound system may be derived from those of the components and from the assembly rules

  → A formal model for composition

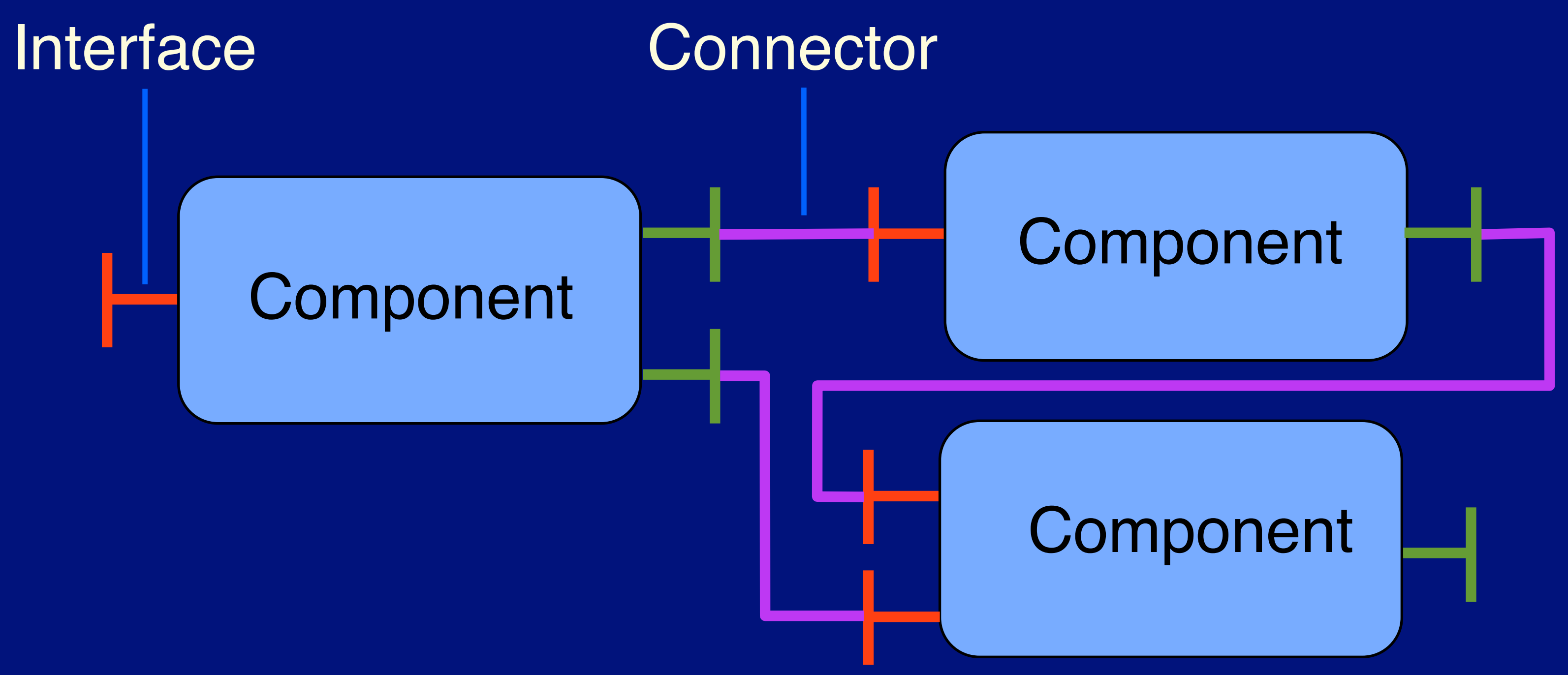
  Expressing semantics

  Run-time guarantees

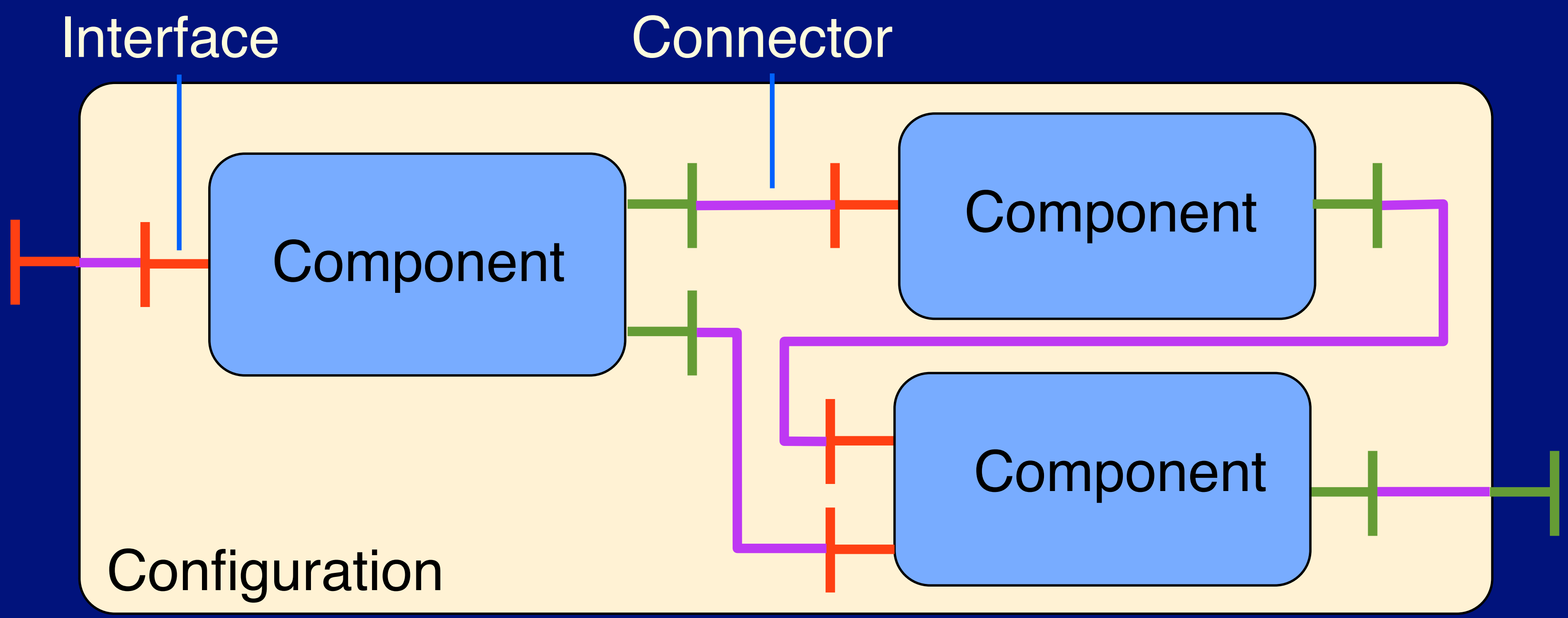
A brief history of (de)composition



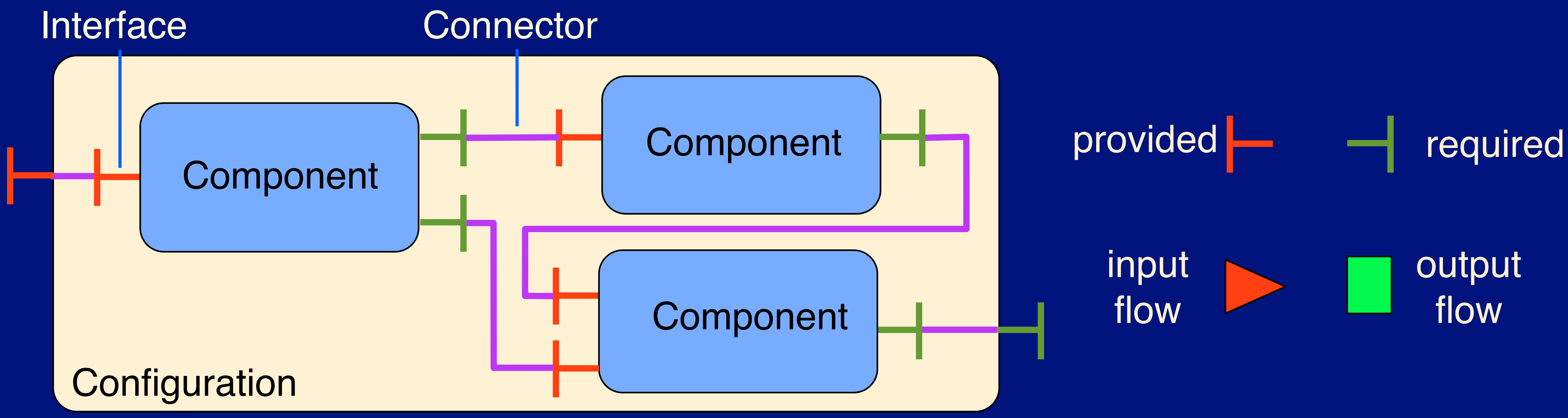
Architecture of compound systems



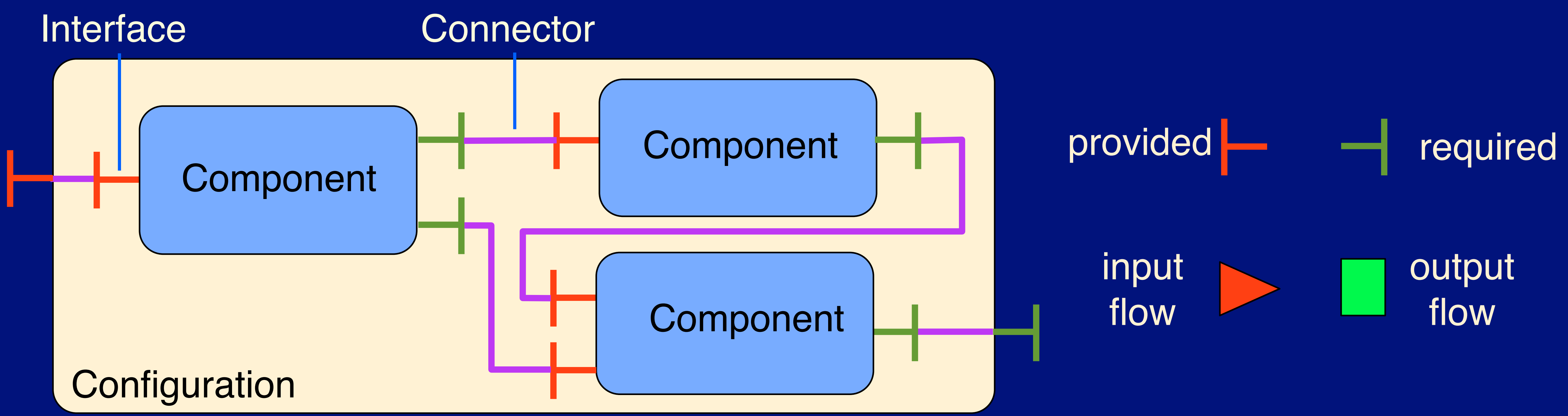
Architecture of compound systems



Architecture of compound systems

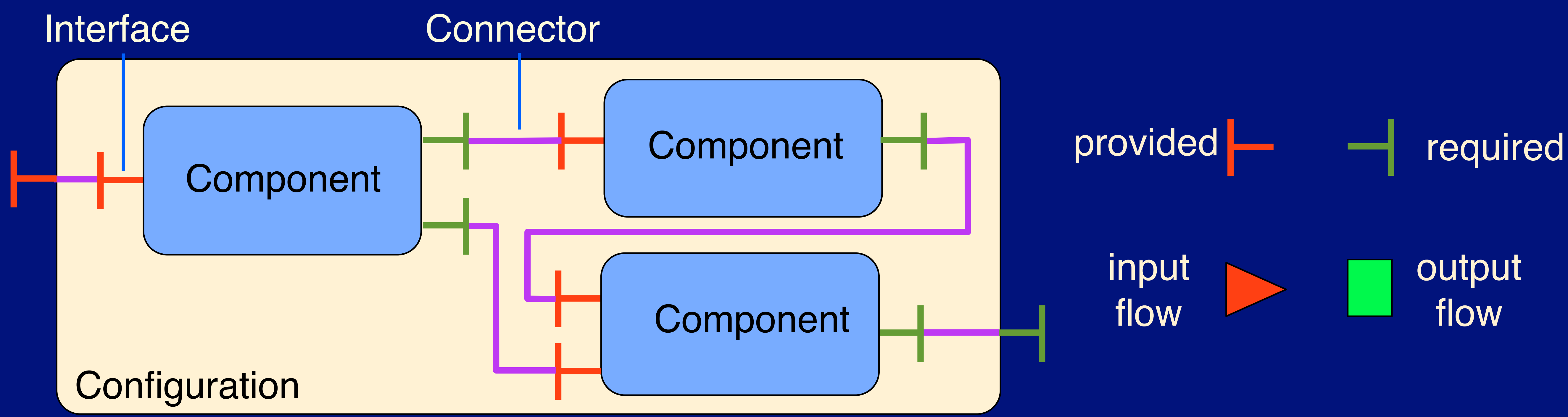


Architecture of compound systems



- Execution flow or data flow
  - Similarities
    - Hardware or software components
    - A configuration is a component
    - Interfaces are typed
  - Differences
    - Role of interfaces or ports
    - Interaction models (sequential, parallel)
      - Synchronous, rendez-vous, events, etc.

Architecture of compound systems



- Execution flow or data flow
  - Similarities
    - Hardware or software components
    - A configuration is a component
    - Interfaces are typed
  - Differences
    - Role of interfaces or ports
    - Interaction models (sequential, parallel)
      - Synchronous, rendez-vous, events, etc.

Global description
- Implicit (dependencies)
- Explicit (*Architecture Description Language, ADL*)

Role of connectors
- Perform binding
  - A complex operation in distributed systems
- Manage interaction
  - Specially in parallel models

Reconfiguration

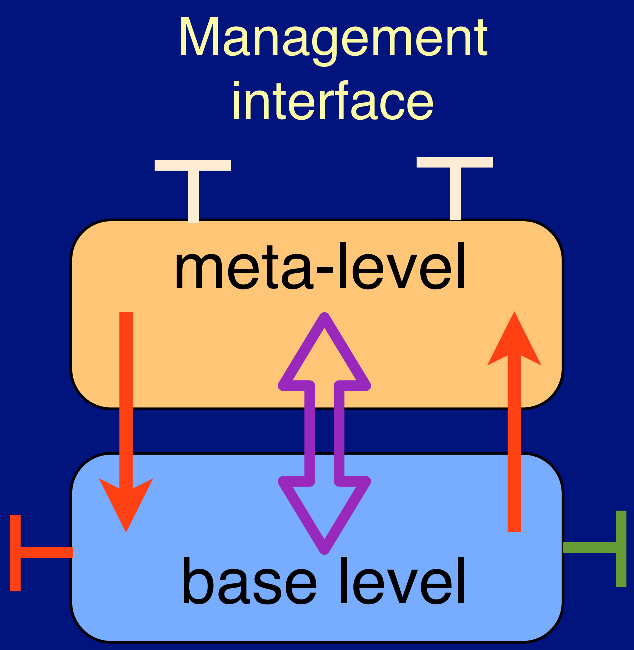
- What is (*dynamic*) reconfiguration?
  - Changing the composition and/or structure of a system *at run time*
    - add/remove a component, move a component, change bindings, modify attributes, …

Reconfiguration

- What is (*dynamic*) reconfiguration?
  - Changing the composition and/or structure of a system *at run time*
    - add/remove a component, move a component, change bindings, modify attributes, …
- Why reconfiguration?
  - Maintenance, optimization, inserting probes for measurement, reacting to failures, overload, attacks, …
  - A natural operation for mobile devices, sensor networks, etc.

Reconfiguration

- What is (*dynamic*) reconfiguration?
  - Changing the composition and/or structure of a system *at run time*
    - add/remove a component, move a component, change bindings, modify attributes, …
- Why reconfiguration?
  - Maintenance, optimization, inserting probes for measurement, reacting to failures, overload, attacks, …
  - A natural operation for mobile devices, sensor networks, etc.
- Good practice
  - Architecture-driven reconfiguration
    - Using reflection
  - Consistency management
    - Preserving invariants
  - Minimal perturbation



Formalizing composition: three examples

- Check the validity of the *construction* of a compound system (composability)

  Reference: http://www.edos-project.org/

  Configuration of an assembly of components

- Check the validity of the *execution* of a compound system (compositionality)

  Application of typing rules

- Check the validity of the *evolution* of a compound system

  Reconfiguration

  M. Léger, Th. Ledoux, Th. Coupaye. Reliable Dynamic Reconfigurations in a Reflective Component Model, *Proc. CBSE 2010*, LNCS 6092, pp. 74-92, Springer Verlag

Compositionality in Dream

Dream: a framework for building communication middleware

A message is a sequence of named fields. Example:

- [Name: "test"] [TS: 10] [IP: 156.875.34.12]

A message transits between components that operate on it

- Typical operations: *add*, *delete*, *consult* a field

Illegal operations (trigger a run-time error)

- *add* an existent field, *delete* or *consult* a non-existent field

M. Leclercq, V. Quéma, J.-B. Stefani, "DREAM: A Component Framework for Constructing Resource-Aware, Configurable Middleware," *Distributed Systems Online, IEEE* , 6:9, Sept. 2005

Compositionality in Dream

Dream: a framework for building communication middleware

A message is a sequence of named fields. Example:

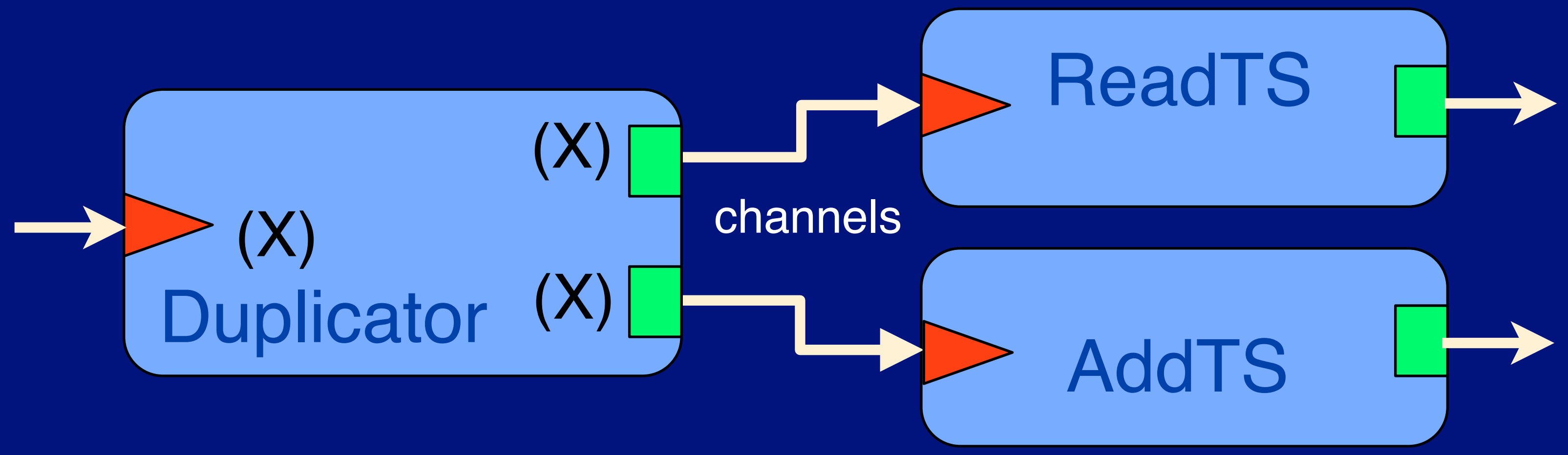
- [Name: "test"] [TS: 10] [IP: 156.875.34.12]

A message transits between components that operate on it

- Typical operations: *add*, *delete*, *consult* a field

Illegal operations (trigger a run-time error)

- *add* an existent field, *delete* or *consult* a non-existent field



M. Leclercq, V. Quéma, J.-B. Stefani, "DREAM: A Component Framework for Constructing Resource-Aware, Configurable Middleware," *Distributed Systems Online, IEEE* , 6:9, Sept. 2005

Compositionality in Dream

Dream: a framework for building communication middleware

A message is a sequence of named fields. Example:

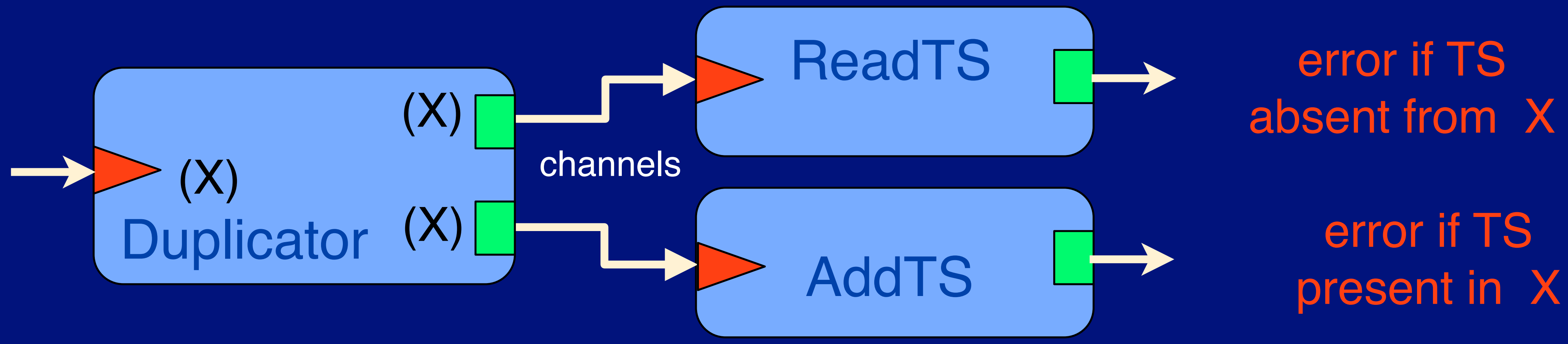
- [Name: "test"] [TS: 10] [IP: 156.875.34.12]

A message transits between components that operate on it

- Typical operations: *add*, *delete*, *consult* a field

Illegal operations (trigger a run-time error)

- *add* an existent field, *delete* or *consult* a non-existent field



M. Leclercq, V. Quéma, J.-B. Stefani, "DREAM: A Component Framework for Constructing Resource-Aware, Configurable Middleware," *Distributed Systems Online, IEEE* , 6:9, Sept. 2005

Compositionality in Dream

Dream: a framework for building communication middleware

A message is a sequence of named fields. Example:

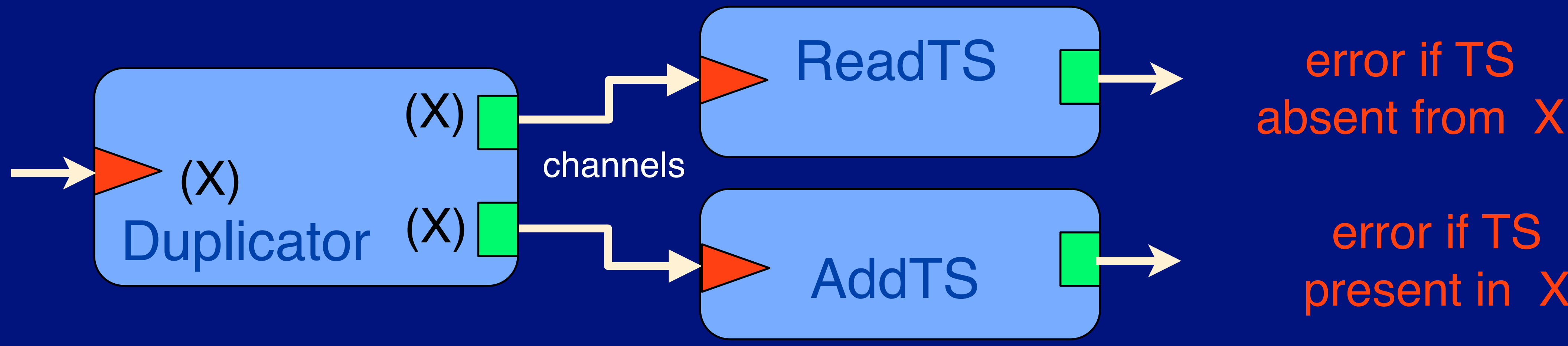
[Name: "test"] [TS: 10] [IP: 156.875.34.12]

A message transits between components that operate on it

Typical operations: *add*, *delete*, *consult* a field

Illegal operations (trigger a run-time error)

add an existent field, *delete* or *consult* a non-existent field

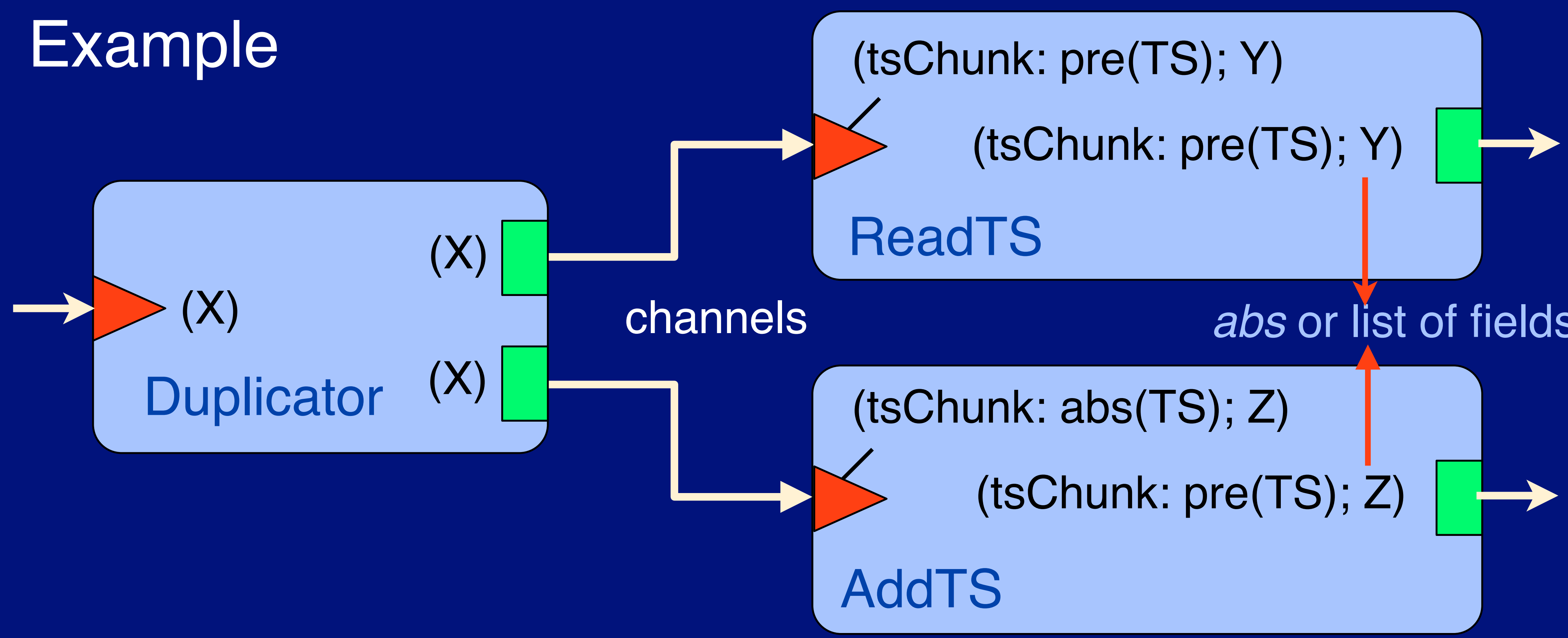


Java types do not allow these checks

M. Leclercq, V. Quéma, J.-B. Stefani, "DREAM: A Component Framework for Constructing Resource-Aware, Configurable Middleware," *Distributed Systems Online, IEEE* , 6:9, Sept. 2005

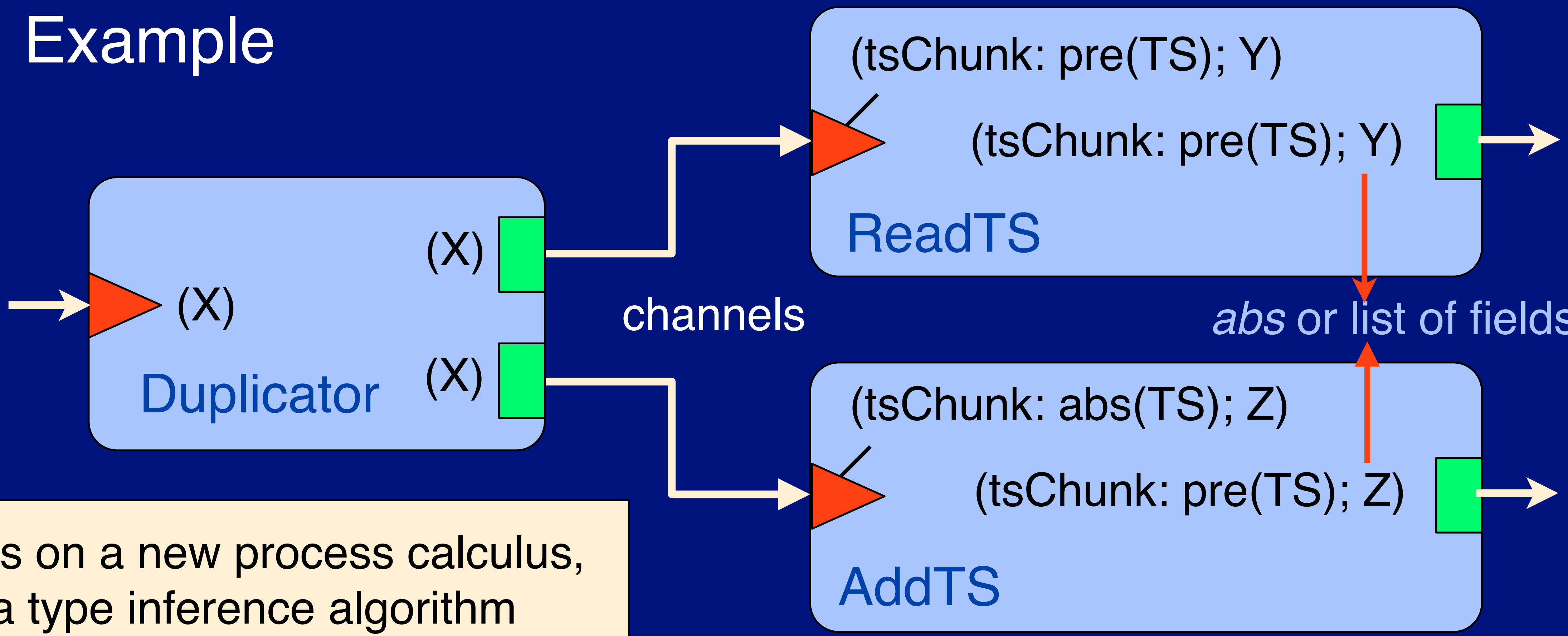
Dream Types

- Example



Dream Types

- Example

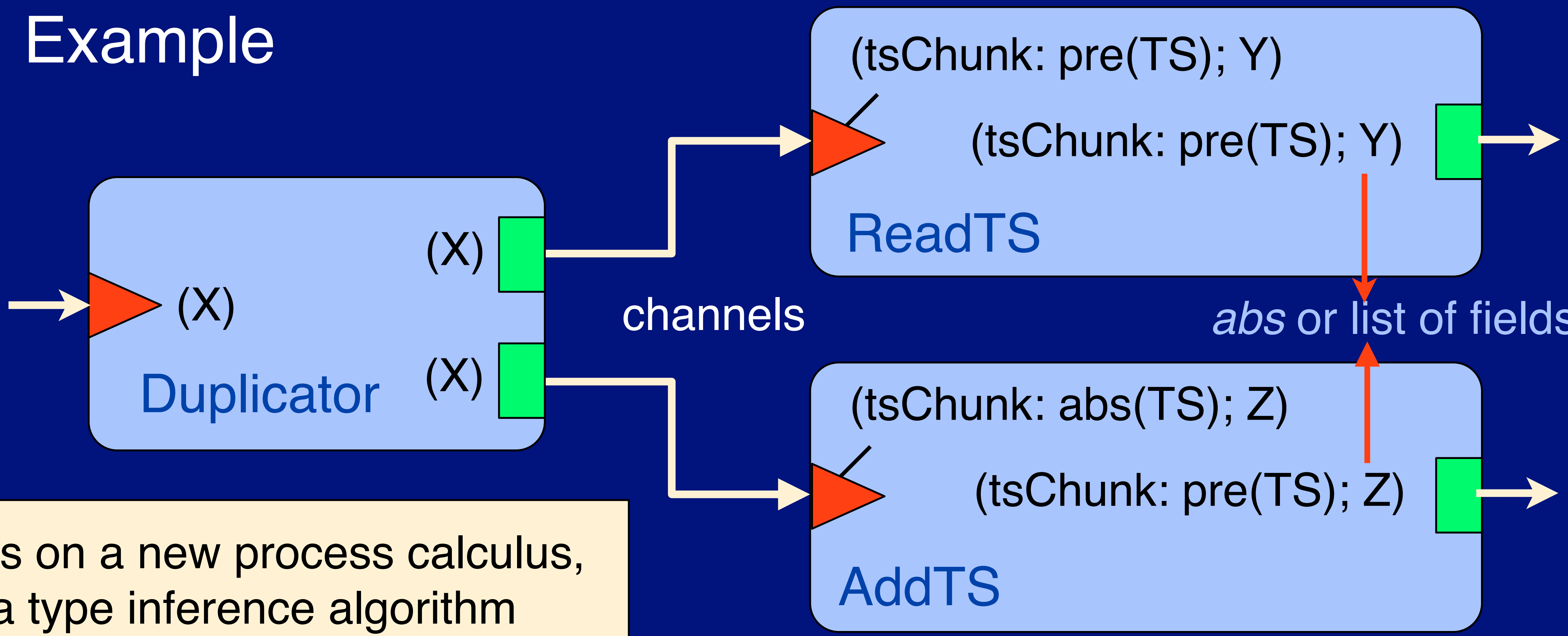


Relies on a new process calculus, with a type inference algorithm

Typing Component-Based Communication Systems. M. Lienhardt, C. A. Mezzina, A. Schmitt, and J.-B. Stefani. In *Proc. 11th Formal Methods for Open Object-Based Distributed Systems (FMOODS) & 29th Formal Techniques for Networked and Distributed Systems (FORTE)*, June 2009.

Dream Types

- Example



Relies on a new process calculus, with a type inference algorithm

Resolve
(X) = (tsChunk: pre(TS); Y)
(X) = (tsChunk: abs(TS); Z)
which is impossible
The system may not be correctly typed, and will fail at run time

Typing Component-Based Communication Systems. M. Lienhardt, C. A. Mezzina, A. Schmitt, and J.-B. Stefani. In *Proc. 11th Formal Methods for Open Object-Based Distributed Systems (FMOODS) & 29th Formal Techniques for Networked and Distributed Systems (FORTE)*, June 2009.

Advances and challenges for composition

- Advances
  - Components, software architectures
  - Patterns and frameworks for composition
  - A step towards formalization

Advances and challenges for composition

- Advances
  - Components, software architectures
  - Patterns and frameworks for composition
  - A step towards formalization
- Challenges
  - Formal bases
  - Multiple models and languages
    - maybe unavoidable …
  - Hardware-software integration
  - Compositionality
    - specially: performance, synchronization, physical time
  - Large scale systems

X

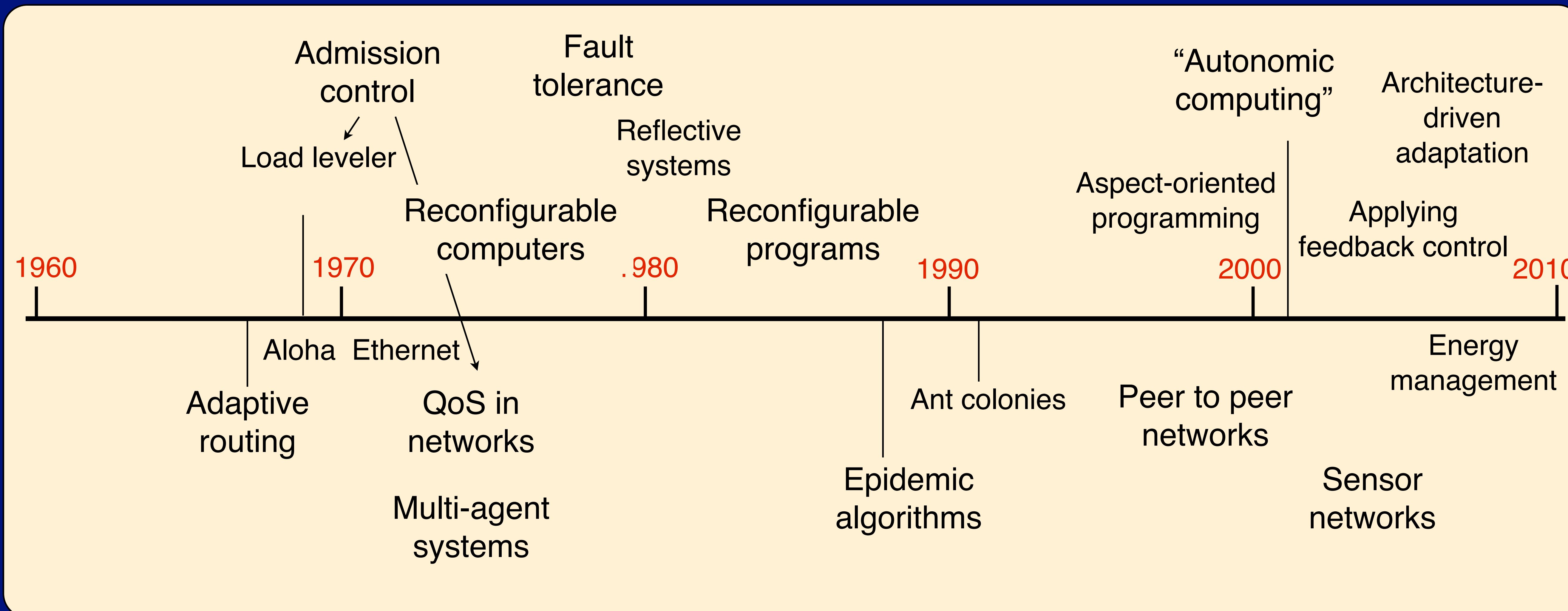
Self-adaptive systems

- Why self-adaptive systems?
  - Preserving integrity and quality of service of a system …
  - … in a changing and unpredictable environment
    - Requirements
    - Load
    - Failures
    - Attacks

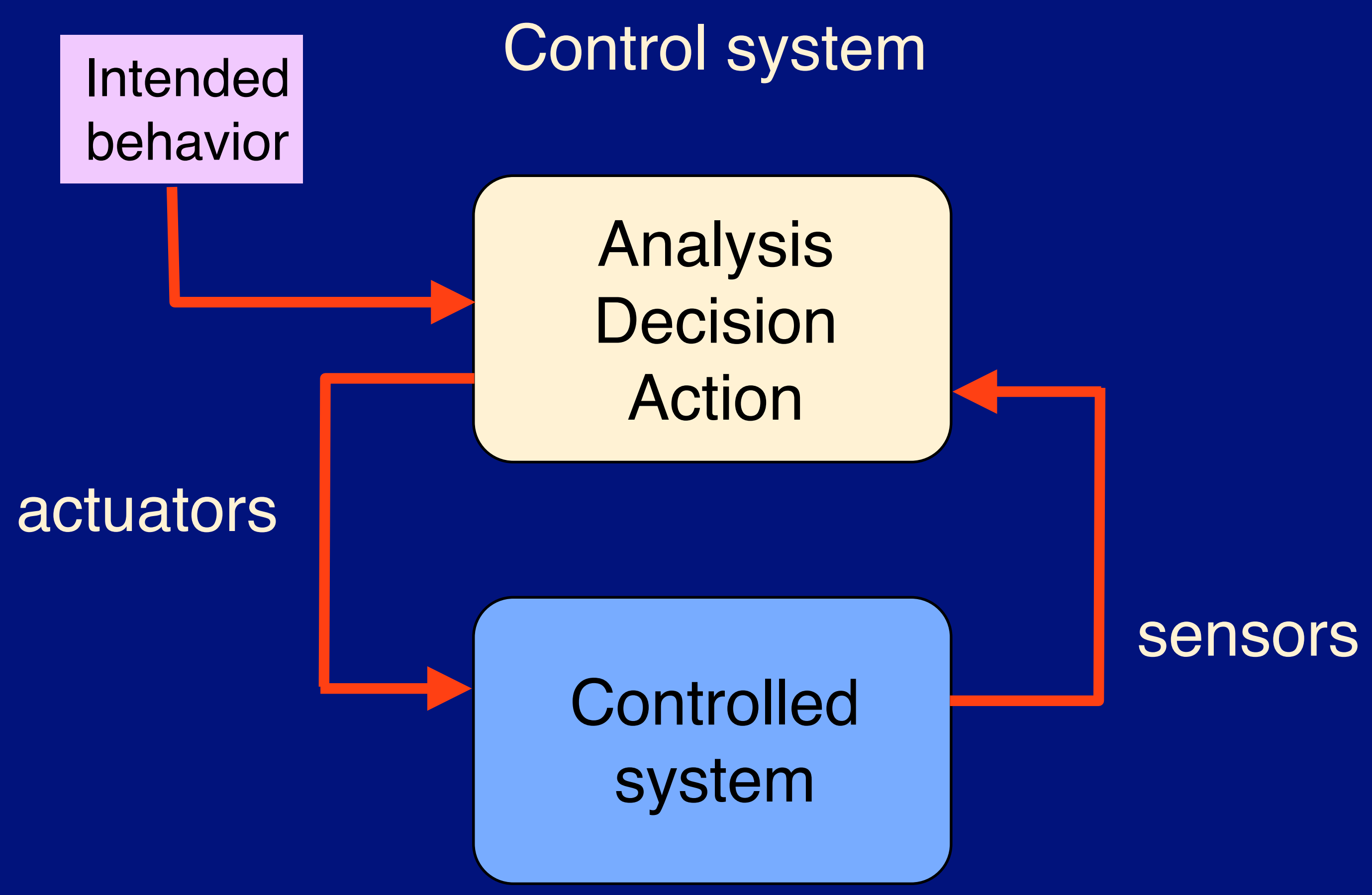
Self-adaptive systems

- Why self-adaptive systems?
  - Preserving integrity and quality of service of a system …
  - … in a changing and unpredictable environment
    - Requirements
    - Load
    - Failures
    - Attacks
- Approaches to self-adaptation
  - Centralized
    - Global behavior is imposed
    - Model: control theory, feedback loop
  - Decentralized
    - Global behavior is determined by local interactions
    - Model: biological systems

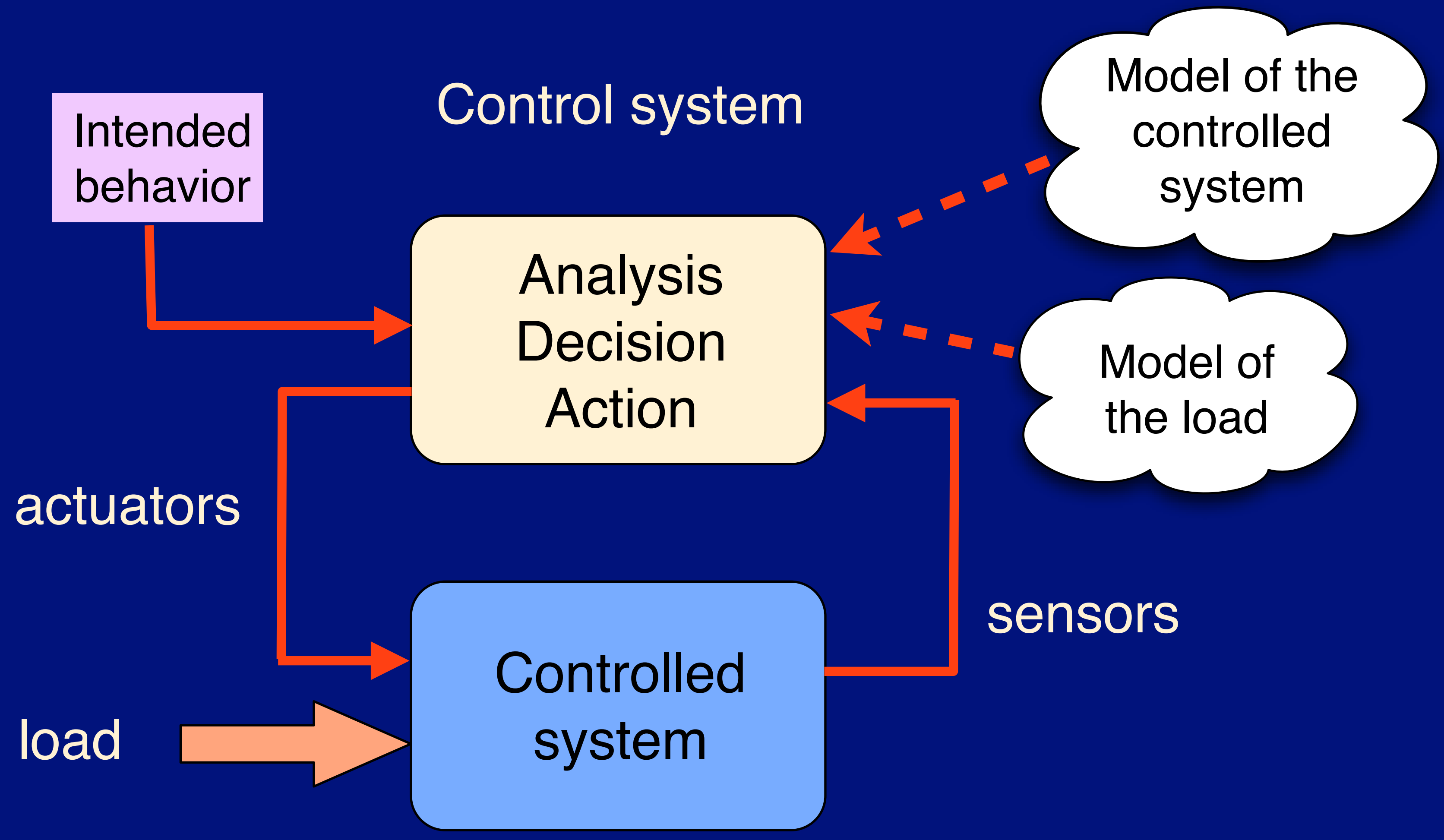
A brief history of self-adaptable computing systems



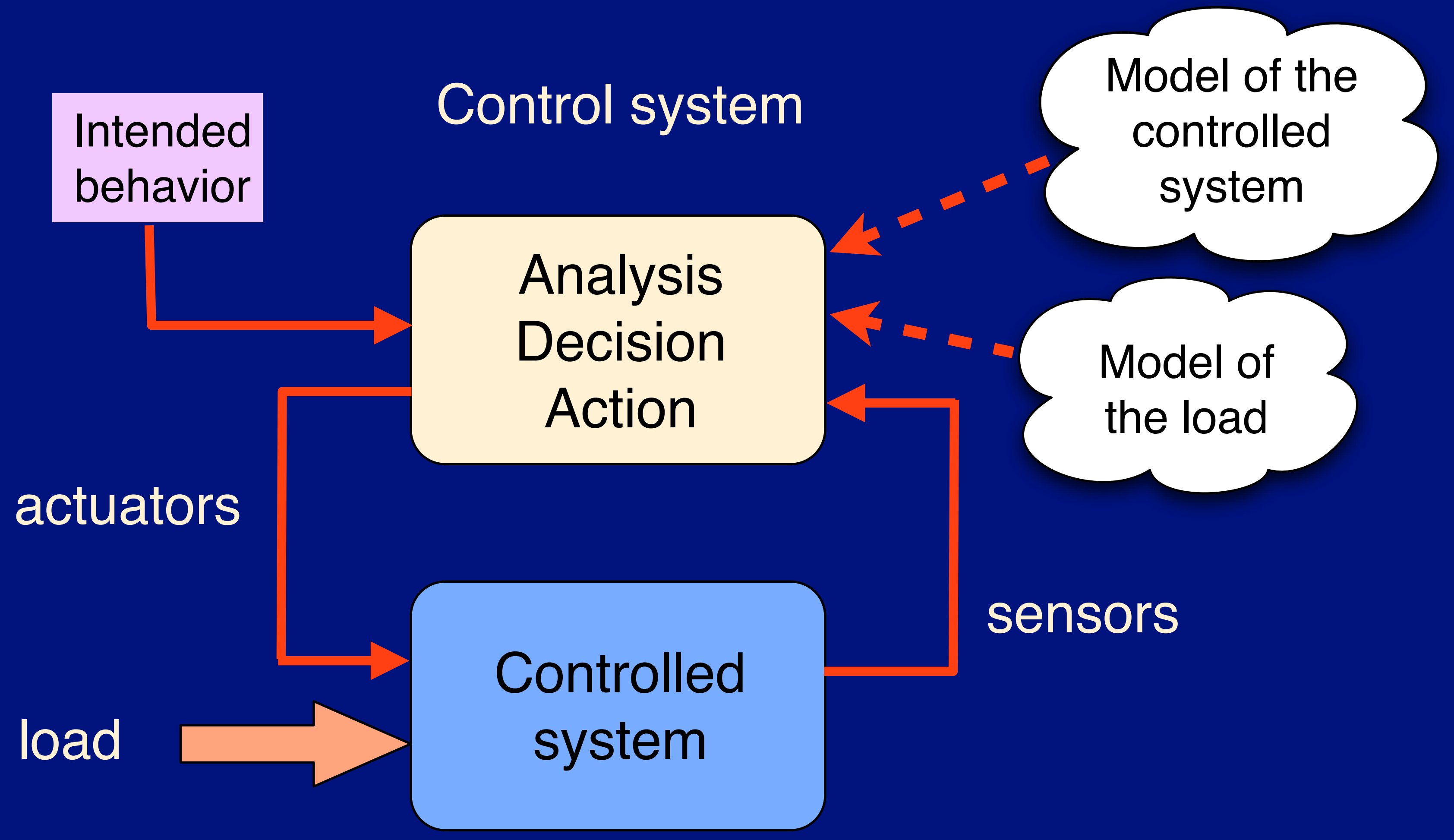
Self-adaptation through feedback



Self-adaptation through feedback



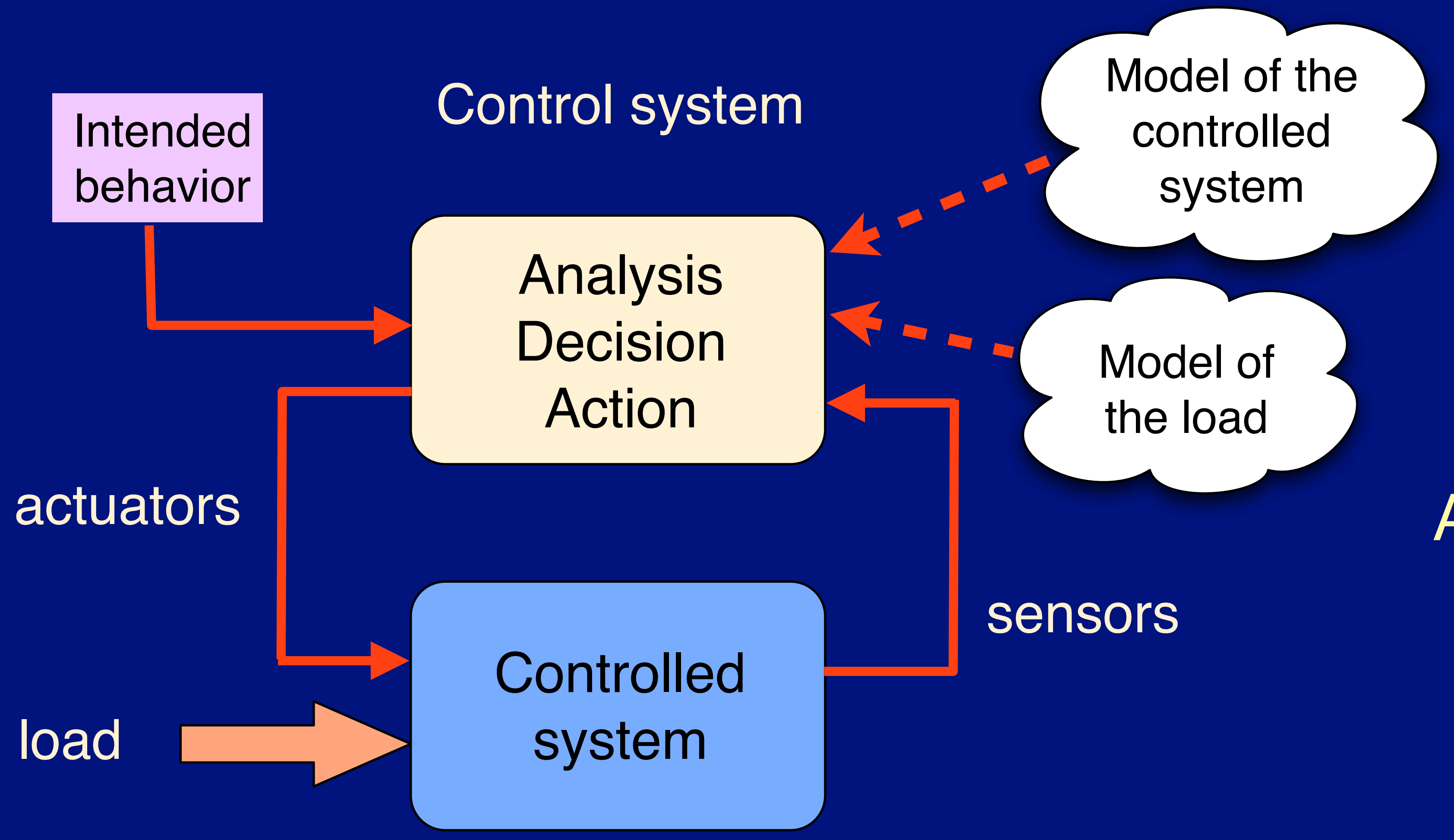
Self-adaptation through feedback



For a computing system, how to define

- The intended behavior?
- The sensors?
- The actuators?
- The models?
- The decision strategy?

Self-adaptation through feedback



Action domains

- Quality of service
- Fault tolerance
- Security
- Configuration

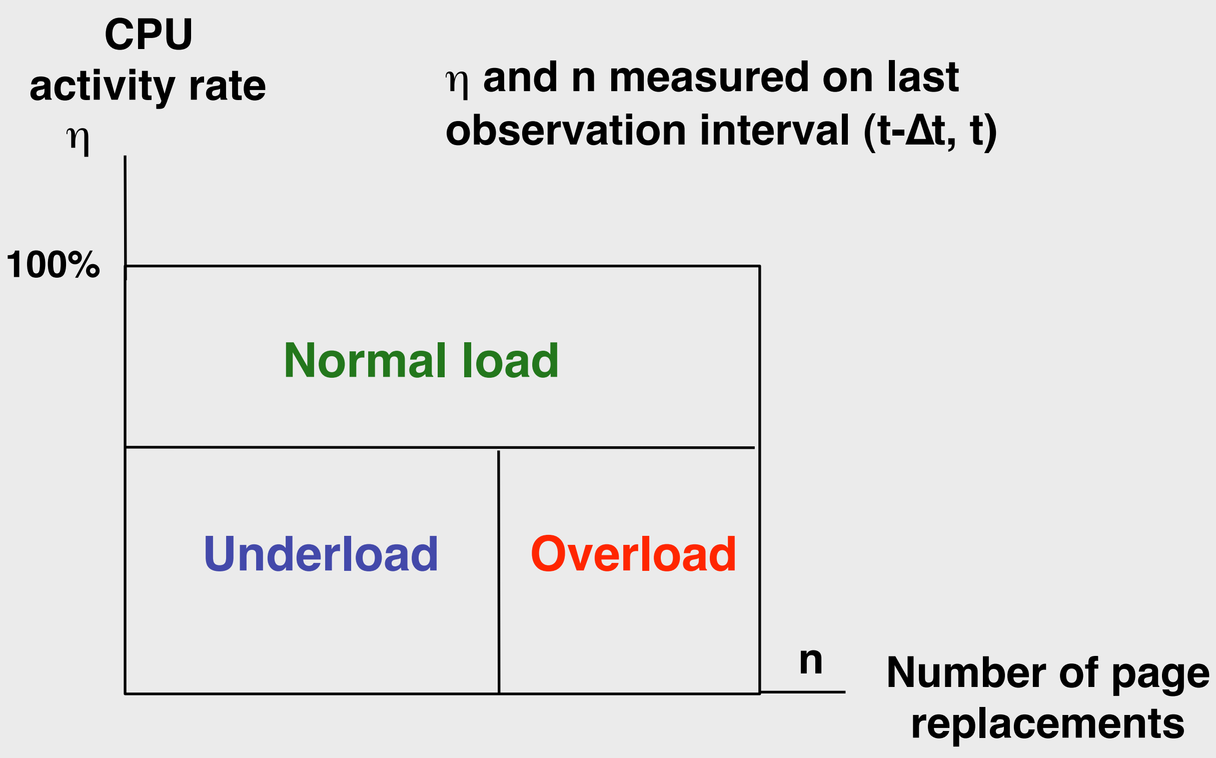
For a computing system, how to define

- The intended behavior?
- The sensors?
- The actuators?
- The models?
- The decision strategy?

A general heuristics

- Admission control

An old example, with admission control

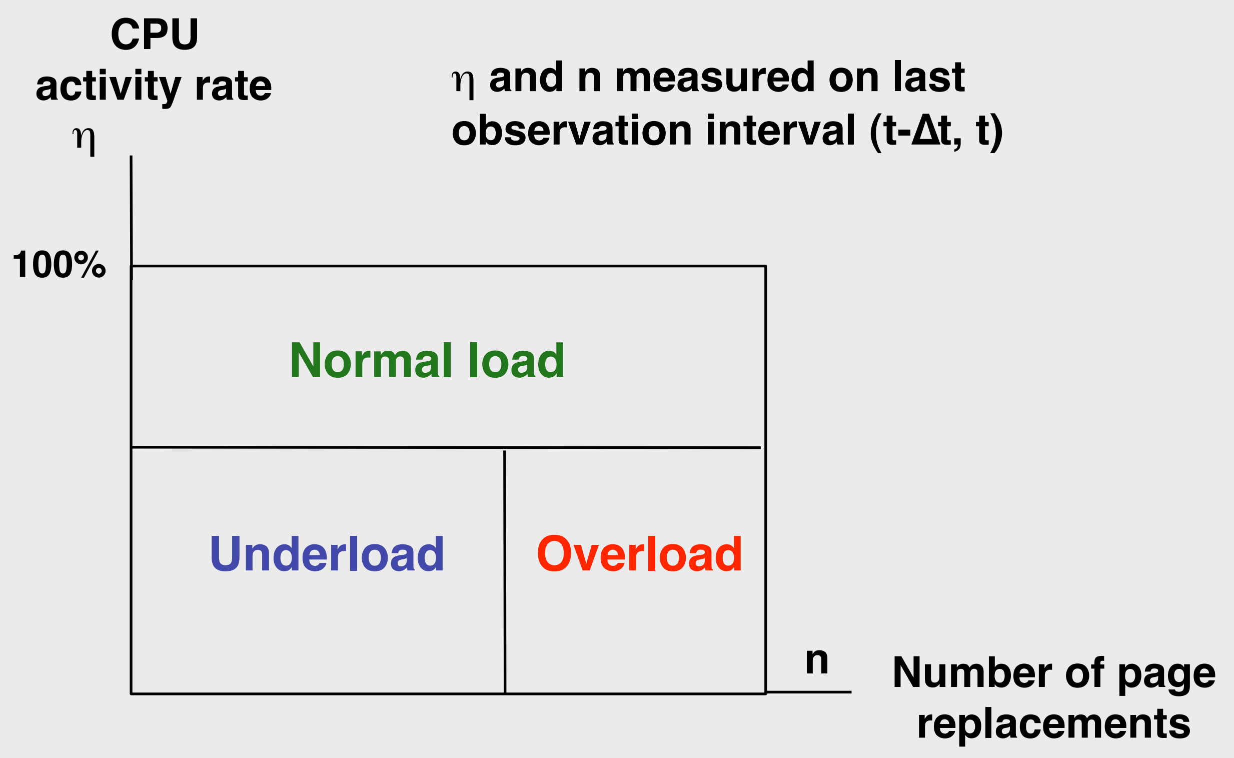


a) a simple model of system behavior

Preventing thrashing:
the IBM M44/44X experiments (1968)

An old example, with admission control

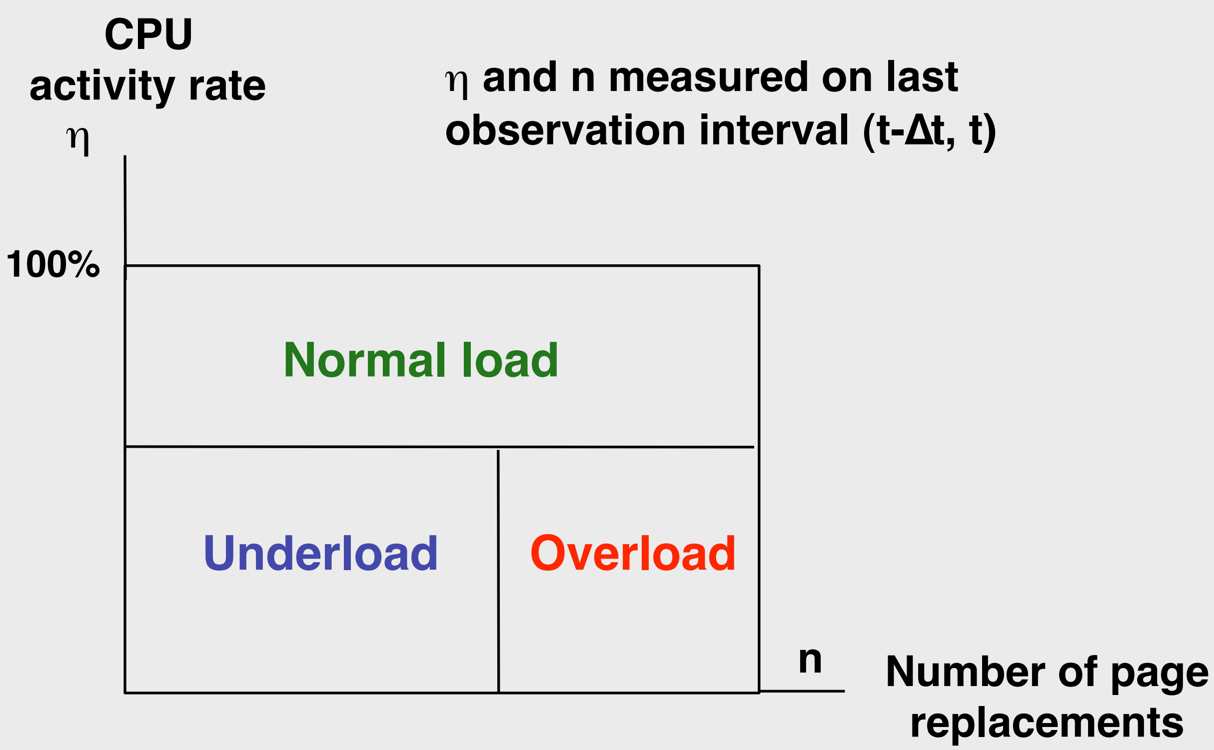
Preventing thrashing: the IBM M44/44X experiments (1968)



a) a simple model of system behavior

```
every Δt do
    if  (overload)
       move one process from ready set to waiting set
    else
       if (underload and (waiting set ≠ ∅))
              admit one waiting process to ready set
```

An old example, with admission control

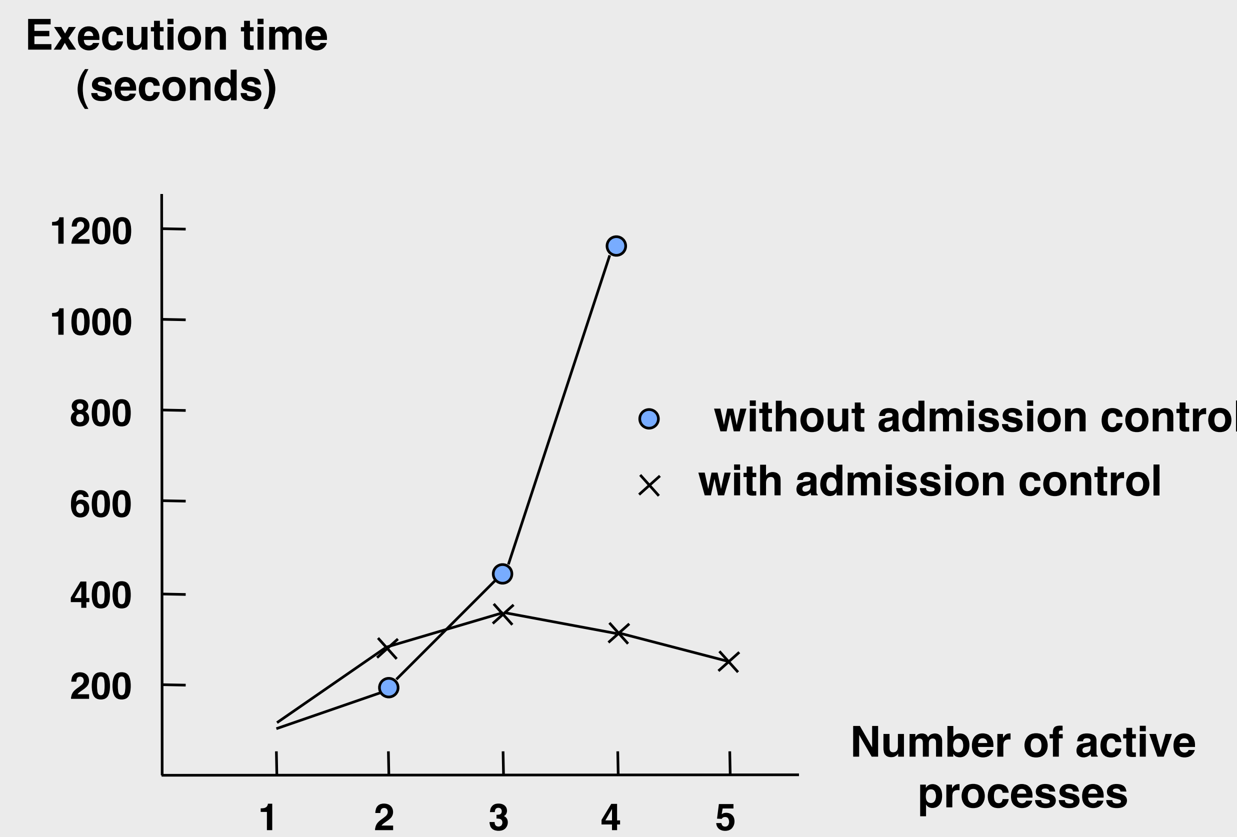


a) a simple model of system behavior

```
every Δt do
    if  (overload)
        move one process from ready set to waiting set
    else
        if (underload and (waiting set ≠ ∅))
                admit one waiting process to ready set
```

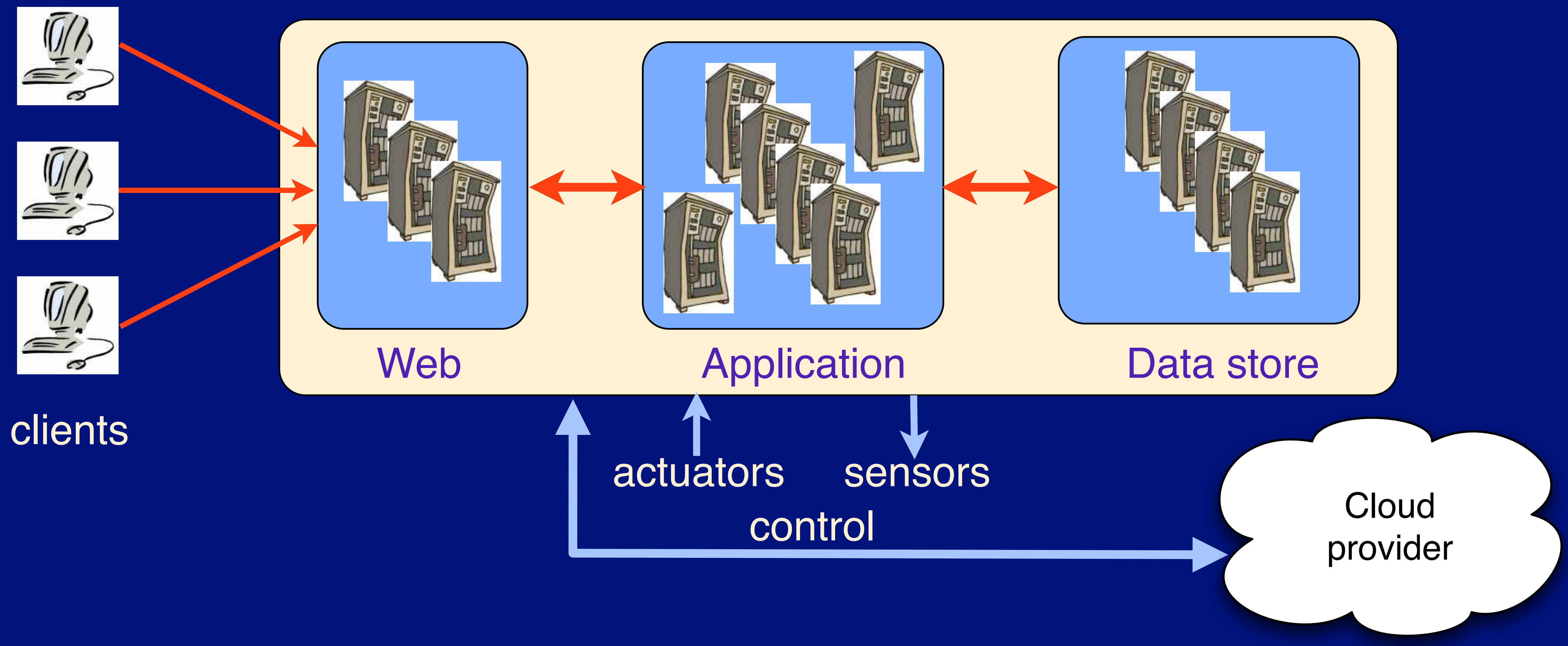
Preventing thrashing: the IBM M44/44X experiments (1968)

B. Brawn, F. Gustavson. Program behavior in a paging environment. *Proc. AFIPS FJCC*, pp. 1019-1032 (1968)

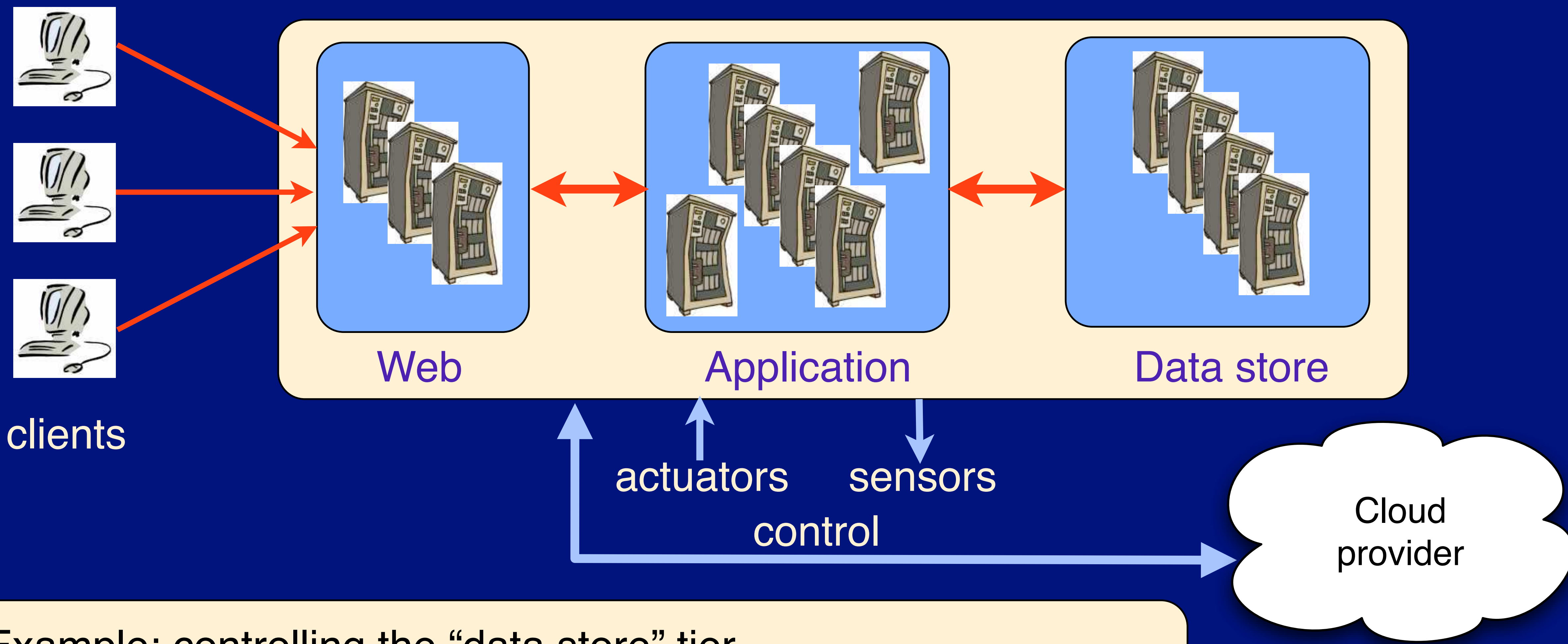


b) effect of load leveling by admission control

Self-adaptation for QoS : example (1)



Self-adaptation for QoS : example (1)



Example: controlling the “data store” tier
- Allocating servers from a cloud provider
- Goal:
  - guaranteeing response time under a bursty load
- Experience with *Hadoop Distributed File System*

H. C. Lim, S. Babu, J. S. Chase. Automated Control for Elastic Storage, *International Conf. On Autonomic Computing (ICAC)*, June 7-11, 2010

Self-adaptation for QoS: example (2)

- Designing control algorithms
  - For server allocation
    - actuator: allocate/free servers (provider interface)
    - sensor: CPU utilization rate (strong correlation with response time)
    - strategy: integral control with threshold (for stability)

Self-adaptation for QoS: example (2)

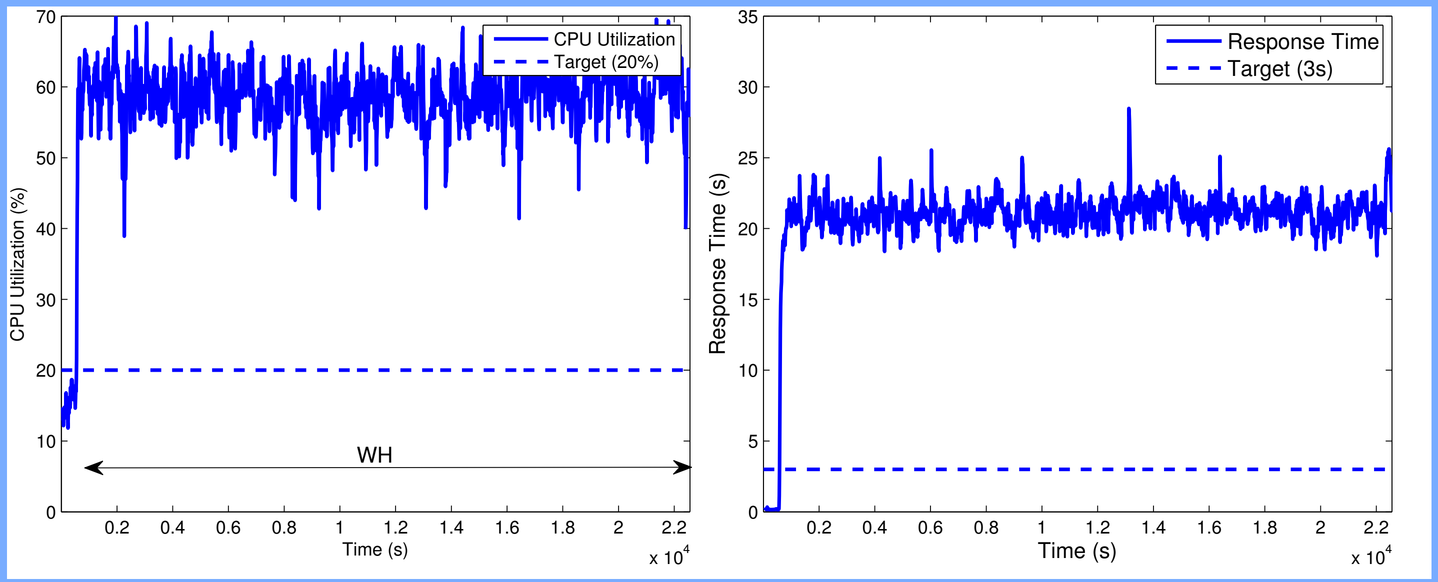
- Designing control algorithms
  - For server allocation
    - actuator: allocate/free servers (provider interface)
    - sensor: CPU utilization rate (strong correlation with response time)
    - strategy: integral control with threshold (for stability)
  - For data store tier reconfiguration (redistributing data)
    - actuator: fraction of bandwidth allocated to reconfiguration (which interferes with request processing)
    - sensor: time needed (a function of data size) + impact of reconfiguration on response time

Self-adaptation for QoS: example (2)

- Designing control algorithms
  - For server allocation
    - actuator: allocate/free servers (provider interface)
    - sensor: CPU utilization rate (strong correlation with response time)
    - strategy: integral control with threshold (for stability)
  - For data store tier reconfiguration (redistributing data)
    - actuator: fraction of bandwidth allocated to reconfiguration (which interferes with request processing)
    - sensor: time needed (a function of data size) + impact of reconfiguration on response time
  - Coordinating the two above control loops
    - goal: avoid over- or under-allocation; avoid oscillations
    - means: state machine ensuring alternation between the two above control loops, with time delay

Self-adaptation for QoS : example (3)

- Results
  - Very good reactivity to a load peak
  - (a posteriori) Good correlation between response time and CPU utilization rate

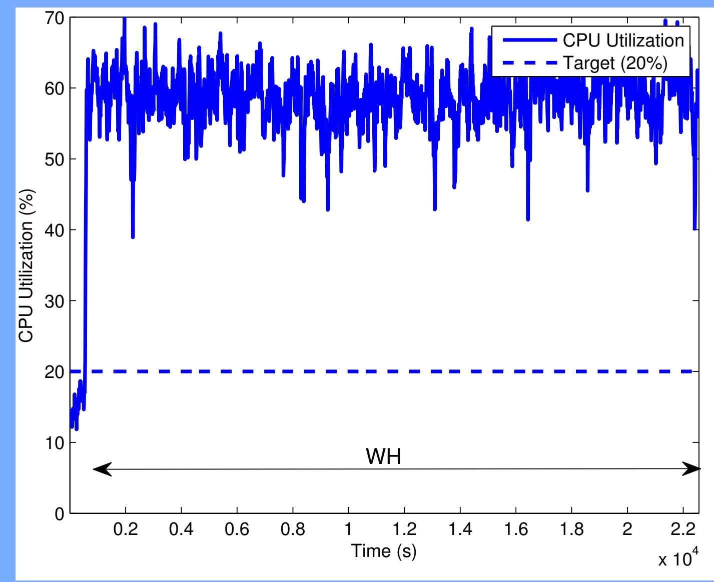


(a) Average CPU utilization of the HDFS datanodes with static provisioning

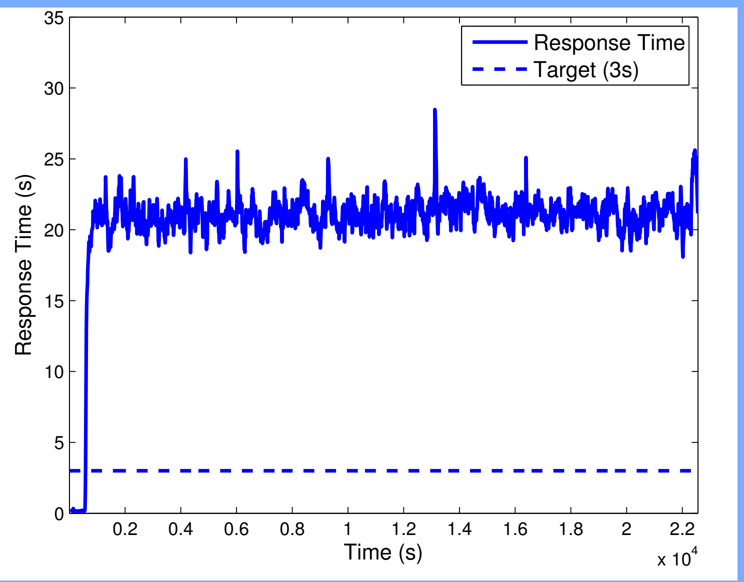
(b) Response time of the Cloudstone application with static provisioning

Self-adaptation for QoS : example (3)

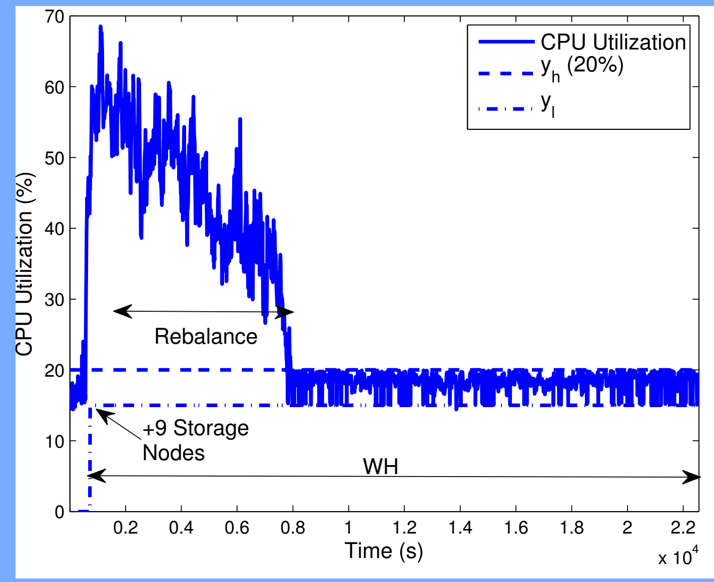
- **Results**
  - Very good reactivity to a load peak
  - (a posteriori) Good correlation between response time and CPU utilization rate



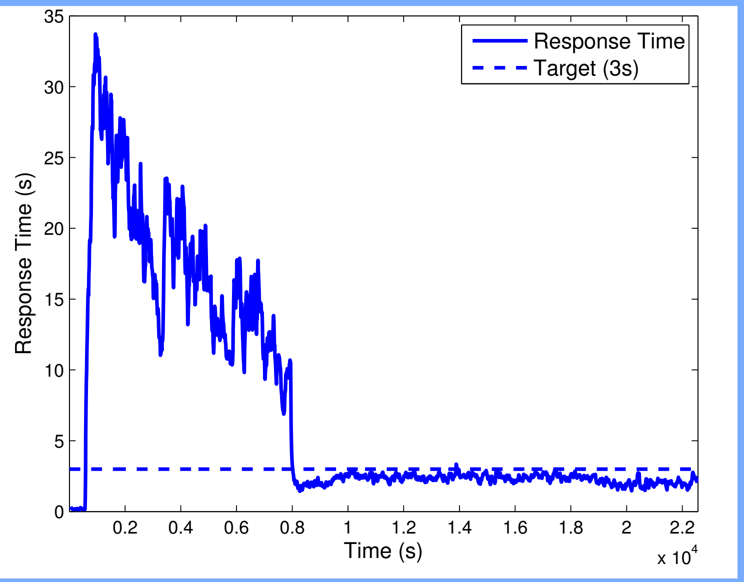
(a) Average CPU utilization of the HDFS datanodes with static provisioning



(b) Response time of the Cloudstone application with static provisioning



(c) Average CPU utilization of the HDFS datanodes with dynamic provisioning



(d) Response time of the Cloudstone application with dynamic provisioning

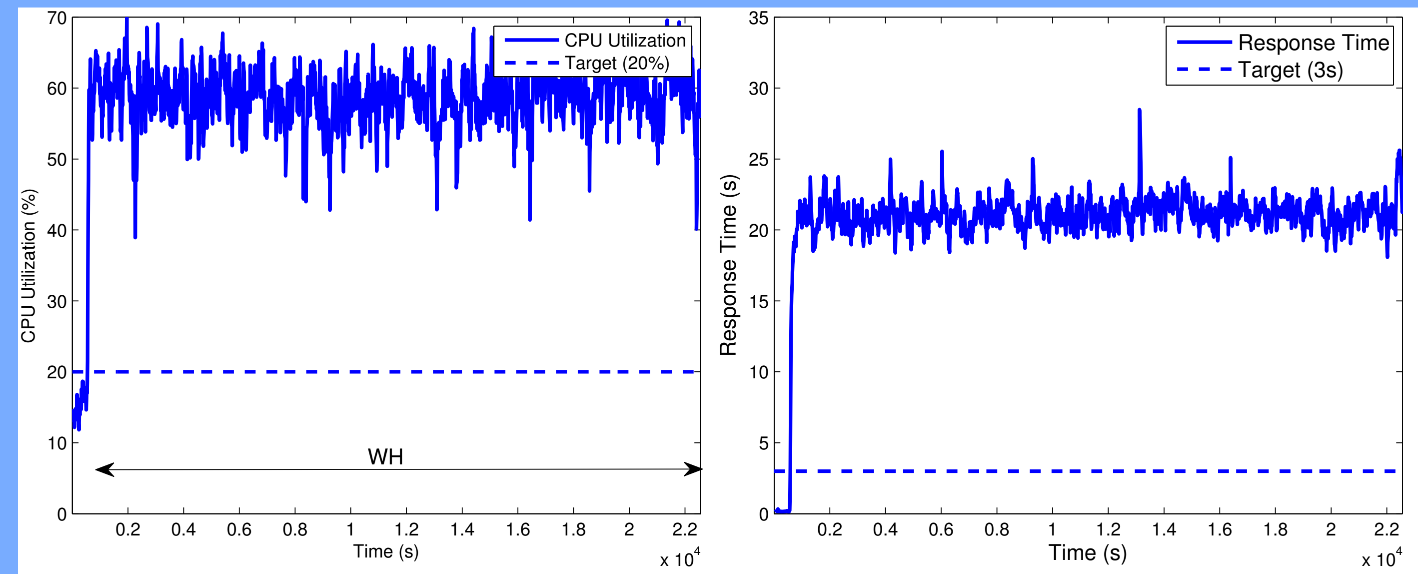
Self-adaptation for QoS : example (3)

- **Results**
  - Very good reactivity to a load peak
  - (a posteriori) Good correlation between response time and CPU utilization rate

Figure 6 of:

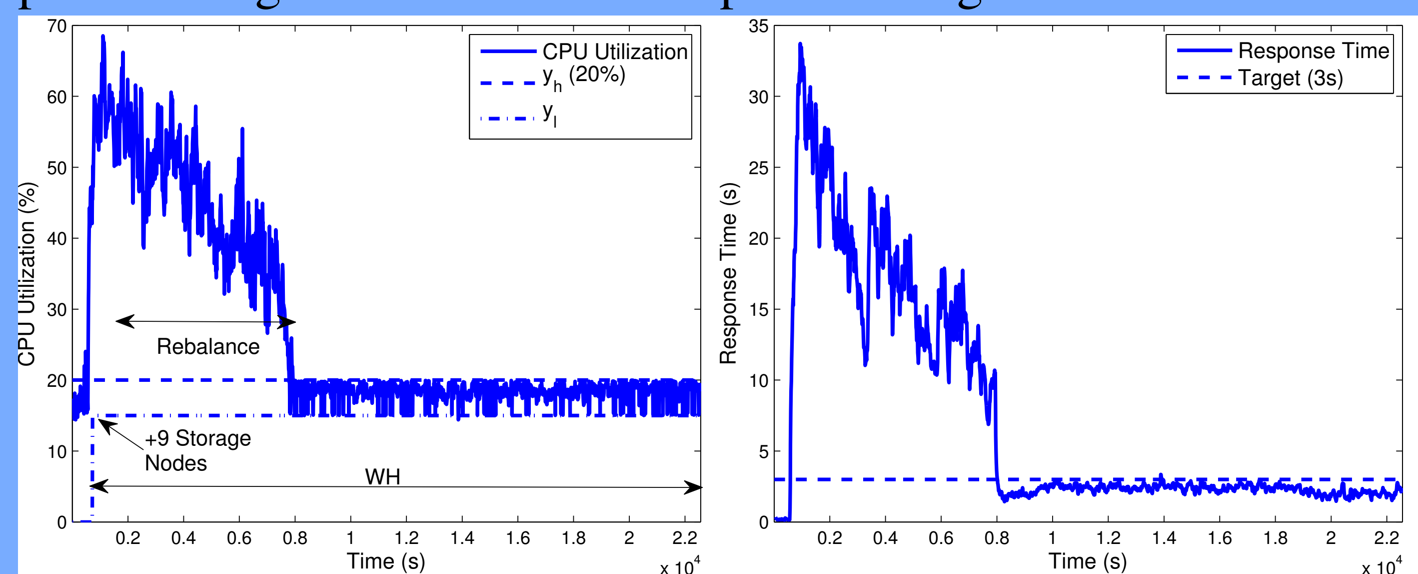
H. C. Lim, S. Babu, J. S. Chase. Automated Control for Elastic Storage, *International Conf. On Autonomic Computing (ICAC)*, June 7-11, 2010

doi>10.1145/1809049.1809052



(a) Average CPU utilization of the HDFS datanodes with static provisioning

(b) Response time of the Cloudstone application with static provisioning



(c) Average CPU utilization of the HDFS datanodes with dynamic provisioning

(d) Response time of the Cloudstone application with dynamic provisioning

Advances and challenges for self-adaptation

- Advances
  - A fruitful interaction with control theory
    - continuous domain (control loop)
    - discrete domain (controller synthesis)
    - some results for QoS
  - Reflective components and architectures

Advances and challenges for self-adaptation

- Advances
  - A fruitful interaction with control theory
    - continuous domain (control loop)
    - discrete domain (controller synthesis)
    - some results for QoS
  - Reflective components and architectures
- Challenges
  - Multilevel approaches (model-driven vs self-organized)
  - Expression of objectives
    - multi-criteria objectives (performance, energy, availability, …)
    - dealing with unexpected situations
  - Modeling, verification, guarantees
    - continuous-discrete interaction, timed models
  - Security

Concluding remarks

- On architectural paradigms
  - Permanence of concepts, (slow) refinement in their application
  - New paradigms
    - mobility, autonomy, …

Concluding remarks

- On architectural paradigms
  - Permanence of concepts, (slow) refinement in their application
  - New paradigms
    - mobility, autonomy, …

> the power of abstraction
>
> the power (and increasing role) of models

Concluding remarks

- On architectural paradigms
  - Permanence of concepts, (slow) refinement in their application
  - New paradigms
    - mobility, autonomy, …
- Some challenges for the future
  - Conceptual
    - formal models for systems architecture
    - validity of construction
    - modeling security
    - hybrid systems
  - Practical
    - declarative description of environments and constraints
    - automatic generation of special-purpose systems
    - administration and quality of service of very large systems

> the power of abstraction
>
> the power (and increasing role) of models

Obrigado pela atenção !