AARHUS UNIVERSITY

DEPARTMENT OF ENGINEERING

Test of Distributed Systems
Lecture 6

From CS to DS:

Going from CS to DS - consequences

Observing DS

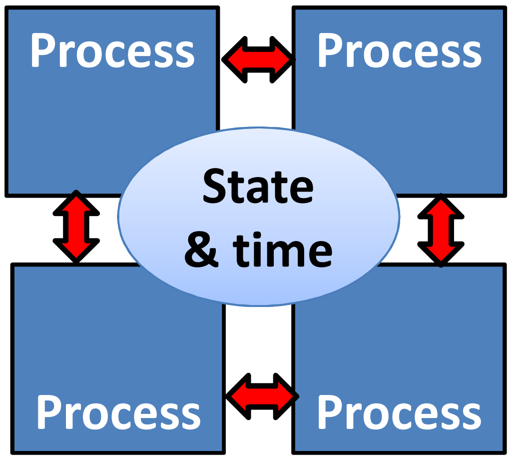
DS properties

Case study – mutual exclusion

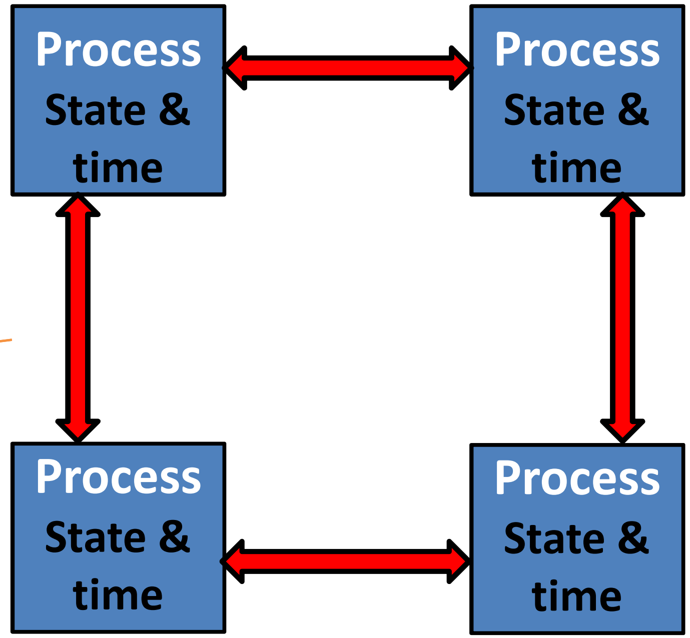
Today’s lecture

- **Going from concurrent systems (CS) to distributed systems (DS)**
- What is meant by DS?
- Concepts: global state, cuts and runs
- Consistency of observed/constructed state
- DS properties – fail safe, fault tolerant
- Case study – mutual exclusion in DS
- Next time

From CoSy to DiSy

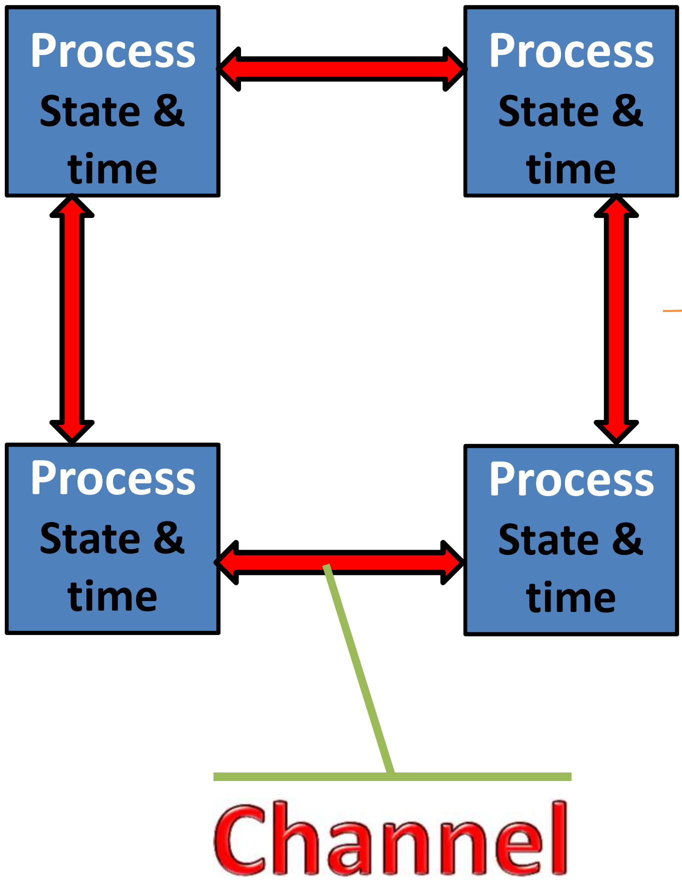


Shared time
Observable global state



No shared time
Global state not observable

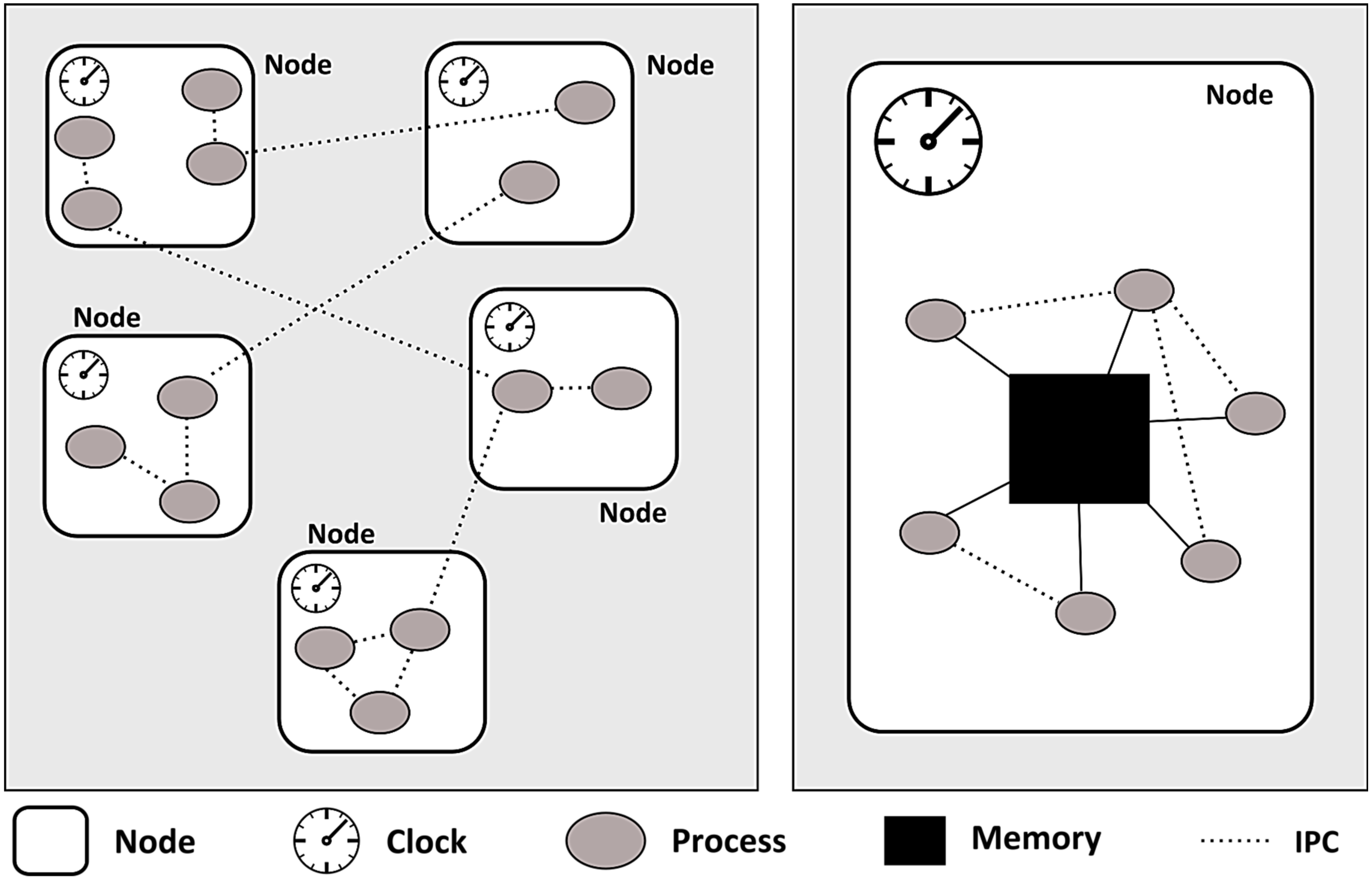
Channels



In distributed systems, the characteristics of the communication channels starts to play a major role:

- Delays
- Reordering
- Lossy or not
- Etc.

Course outline – our model of a distributed system



Non-determinism

- When we loose a shared, common time across processes, another type of non-determinism, or perhaps relativity, occurs
- We cannot observe the system’s state!
- We can only observe the system through message exchanges, and by the time we have a status report from each process, their states may have changed
- The best we can do, is to construct images of global state that *may* have existed

Today’s lecture

- Going from concurrent systems (CS) to distributed systems (DS)
- **What is meant by DS?**
- Concepts: global state, cuts and runs
- Consistency of observed/constructed state
- DS properties – fail safe, fault tolerant
- Case study – mutual exclusion in DS
- Next time

What do we mean by DS?

- Remember: no shared clock / time frame
- Also: no shared memory
- Makes it hard to
  - Reconstruct event ordering
  - Reason about causal relationships
  - Construct global state image in a deterministic manner – 2 different observers may construct different images ("relativistic effect")

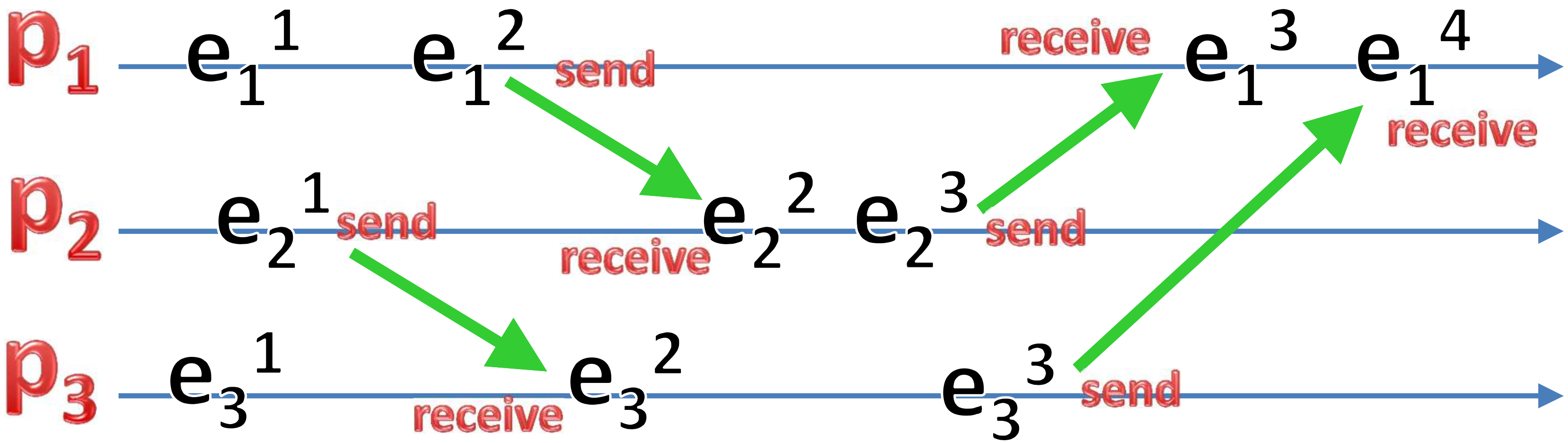
What do we mean by DS?
- model for asynchronous DS

- Collection of sequential ***processes***
  $p_1, p_2, p_3, \ldots, p_n$
- A network of unidirectional ***channels*** between pairs of processes
- Channels are reliable but may deliver messages out-of-order
- ***Asynchronous*** means: no bounds on relative speeds of processes and message delays

What do we mean by DS?
- model for distributed computation

- A collection of sequential processes, each executing a sequence of ***events***
- An event may be *internal* (cause only local state change) or involve *communication* with another process (influence, or be influenced by, external state)
- Communication is accomplished through the events ***send(m)*** and ***receive(m)***
- Send(*m*) = queue *m* at outgoing channel
- Receive(*m*) = dequeue *m* from incoming channel

What do we mean by DS? - model for distributed computation



- ***Local history*** of process $\mathbf{p_i}$ is a (possibly infinite) sequence of events $\mathbf{h_i=e_i^1\ e_i^2\ e_i^3\ \ldots\ e_i^k}$
  Note: History for a single process is a totally ordered sequence of local events(canonical enumeration)

What do we mean by DS?
- ordering events globally

- Global history is the set containing all events

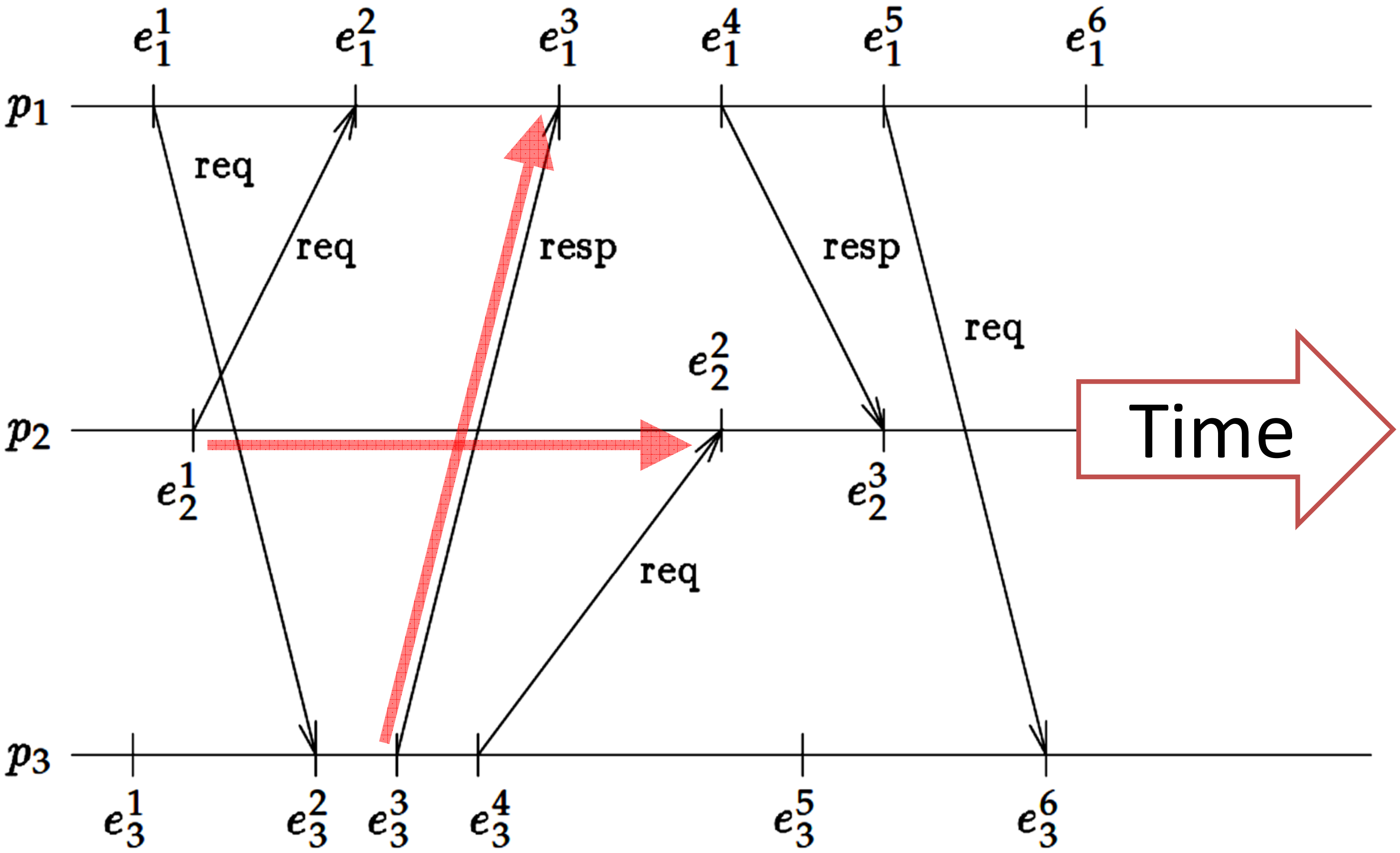
  $$\mathbf{H = h_1 \cup h_2 \cup h_2 \ldots \cup h_d}$$

- *Local* history fully orders events, but *global* history raises problem: how to order events in general (globally)
- In an ADS, with no global time frame, the only useful event ordering is based on "cause-and-effect" (causal relationship)

What do we mean by DS?
- ordering events globally

- Only if occurrence of one event may affect another, are they constrained to occur in a certain order
- In an ADS, an event may affect another (cause a change in the state it sees) if (and only if)
  - they occur in the same process
  - they occur in different processes but they are linked by at least one message exchange

What do we mean by DS?
- events affecting each other



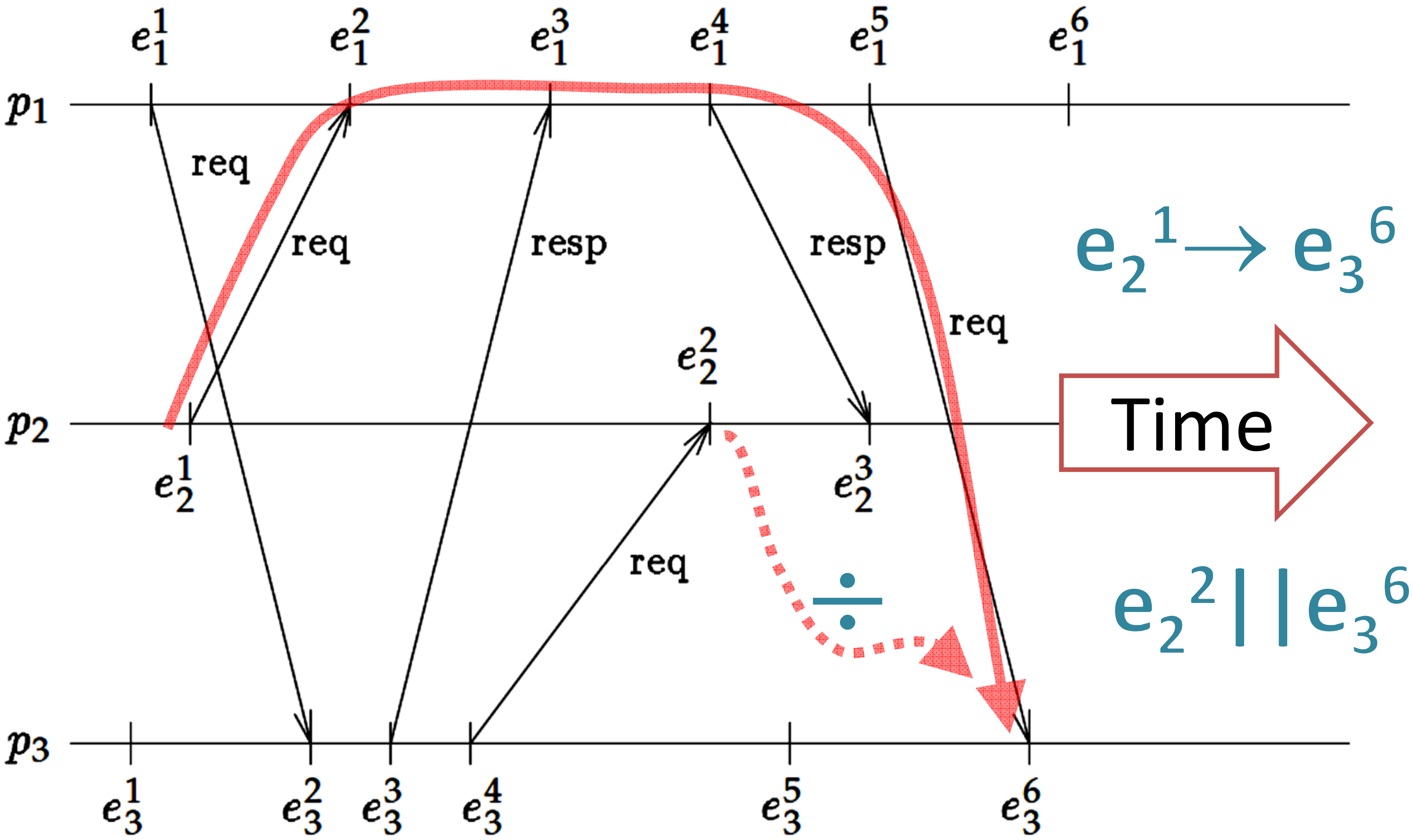
What do we mean by DS? - formalizing global event ordering

- We define a binary relation "$\rightarrow$" = "may causally influence"/"occurs in causal context of"

1. **If $e_i^k, e_i^l \in h_i$, and $k < l$, then $e_i^k \rightarrow e_i^l$**
2. **If e = send(m) and e'=receive(m), then $e \rightarrow e'$**
3. **If $e \rightarrow e'$ and $e' \rightarrow e''$, then $e \rightarrow e''$**

- In global history some events e and e' may not be related, i.e. *not(e $\rightarrow$ e') and not(e' $\rightarrow$ e)*
  - such events are ***concurrent*** and written **e||e'**

What do we mean by DS?
- causal relations



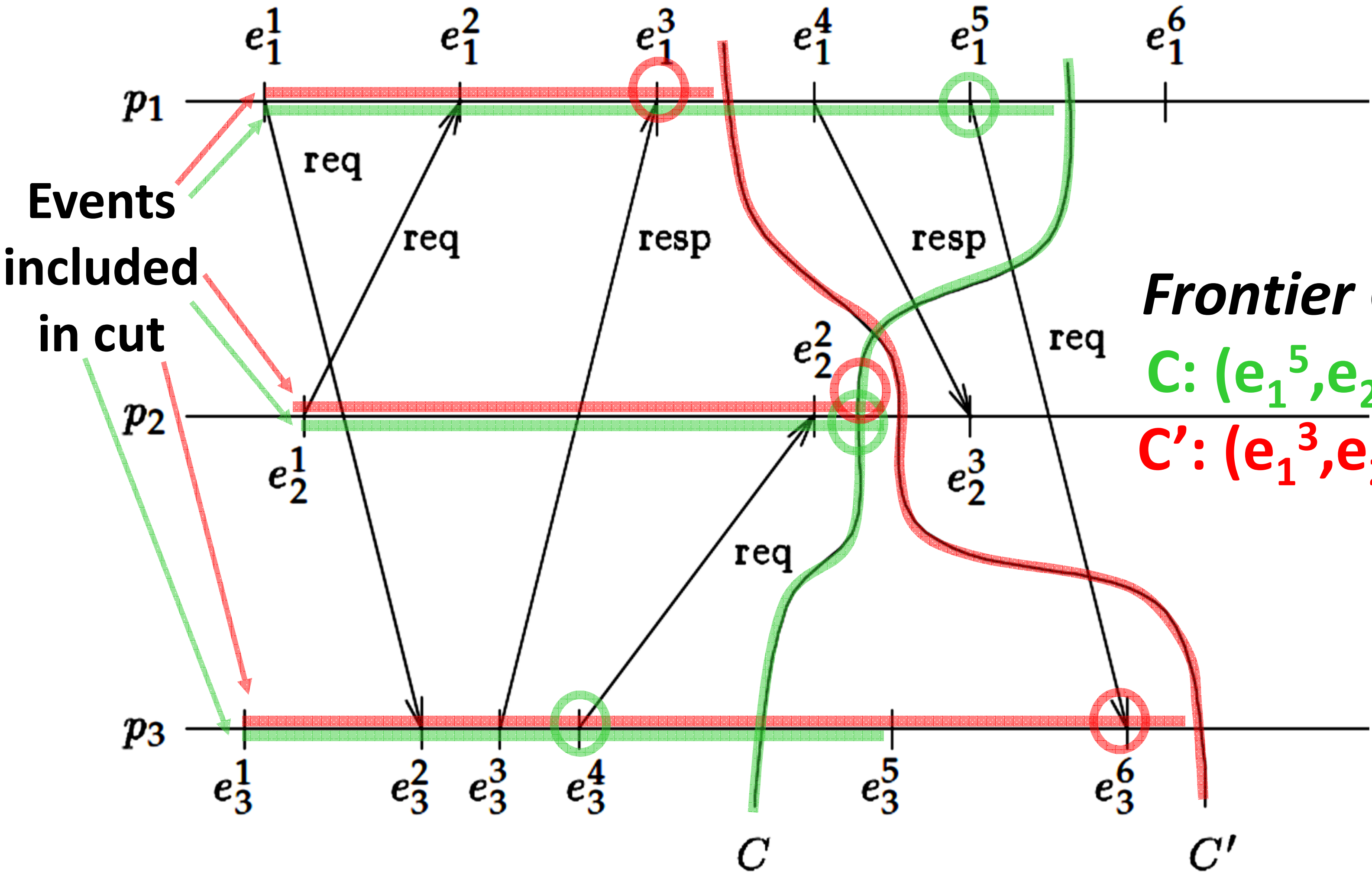
Today’s lecture

- Going from concurrent systems (CS) to distributed systems (DS)
- What is meant by DS?
- **Concepts: global state, cuts and runs**
- Consistency of observed/constructed state
- DS properties – fail safe, fault tolerant
- Case study – mutual exclusion in DS
- Next time

Global state - n-tuple of local states

- **Local state:**
  σ_i^k = local state of process $\mathbf{p}_i$ after event $\mathbf{e}_i^k$
  σ_i^0= initial state of process $\mathbf{p}_i$
- Local state info: values of variables, sequences of messages sent/received
- **Global state:**
  n-tuple of local states, $\Sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$

Cut of a distributed computations - a subset of global history



Frontier of cut

C: (e_1^5, e_2^2, e_3^4)

C': (e_1^3, e_2^2, e_3^6)

Distributed computations - different runs for same calculation

- Local history presents a totally ordered set of events
- Global history is only partially ordered, based on causal relations
- In actual systems, all events occur in some total order (we just can't observe it)
- To be able to reason about executions in DSs, we use the notion of a ***run***

Distributed computations - different runs

- A ***run*** of a DS is a total ordering R that includes all events in global history H, and preserves ordering of local histories (an "interleaving")
- A run may not correspond to any physical execution (impossible interleaving)
- A single distributed computation may have many runs, each corresponding to a different execution (many valid interleavings)

Today’s lecture

- Going from concurrent systems (CS) to distributed systems (DS)
- What is meant by DS?
- Concepts: global state, cuts and runs
- **Consistency of observed/constructed state**
- DS properties – fail safe, fault tolerant
- Case study – mutual exclusion in DS
- Next time

Consistency of observed/constructed state

- What we need is a method/formalism to help us distinguish between cuts that are consistent and those that are not
- Our causal relation $\rightarrow$ turns out to be just the thing. A cut C is ***consistent*** if
  $\forall$ **events e, e': (e $\in$ C) $\wedge$ (e'$\rightarrow$ e) $\Rightarrow$ e' $\in$ C**
- I.e: If e is in our cut C, and e may be causally affected be e', e' *must also* be included in C. So a consistent cut is left closed under the causal precedence relation.

Consistency of observed/constructed state

- A consistent cut corresponds to an "instant" or a "situation" that *may* actually have occurred
- Only global states constructed from consistent cuts (consistent global states) are consistent
- Predicate values are only meaningful when evaluated in the context of a consistent global state

Consistency of observed/constructed state

- A **run *R*** is *consistent* if
  $\forall$ events e, e' where $e \rightarrow e'$,
  e appears before e' in R
- I.e: the total order imposed by a consistent run R must not only preserve local history order, but also the partial order defined by causal precedence.

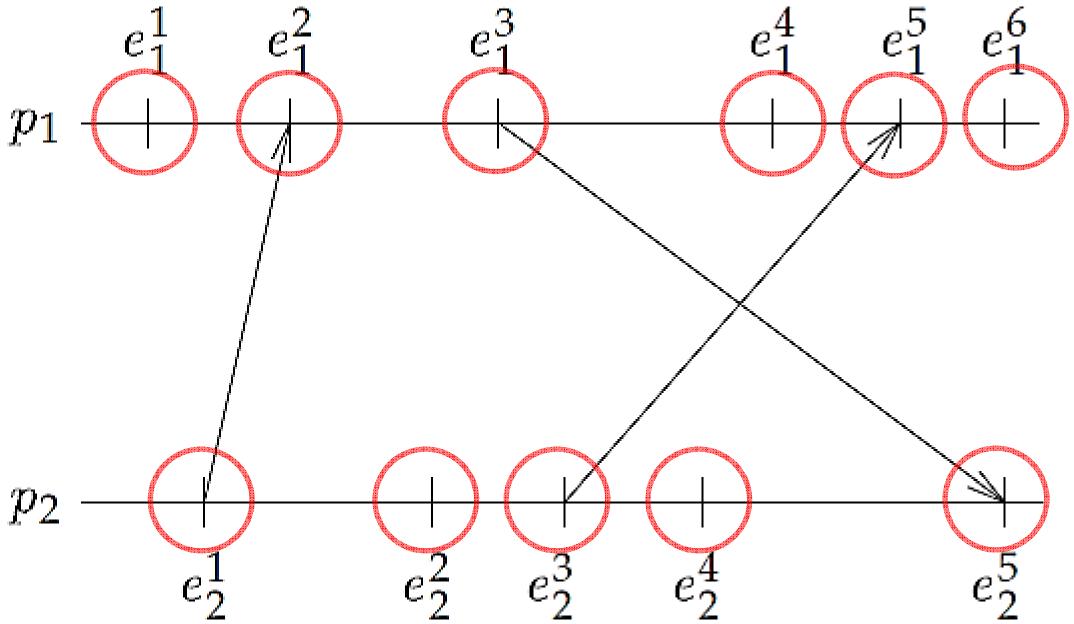
Consistency of observed/constructed state

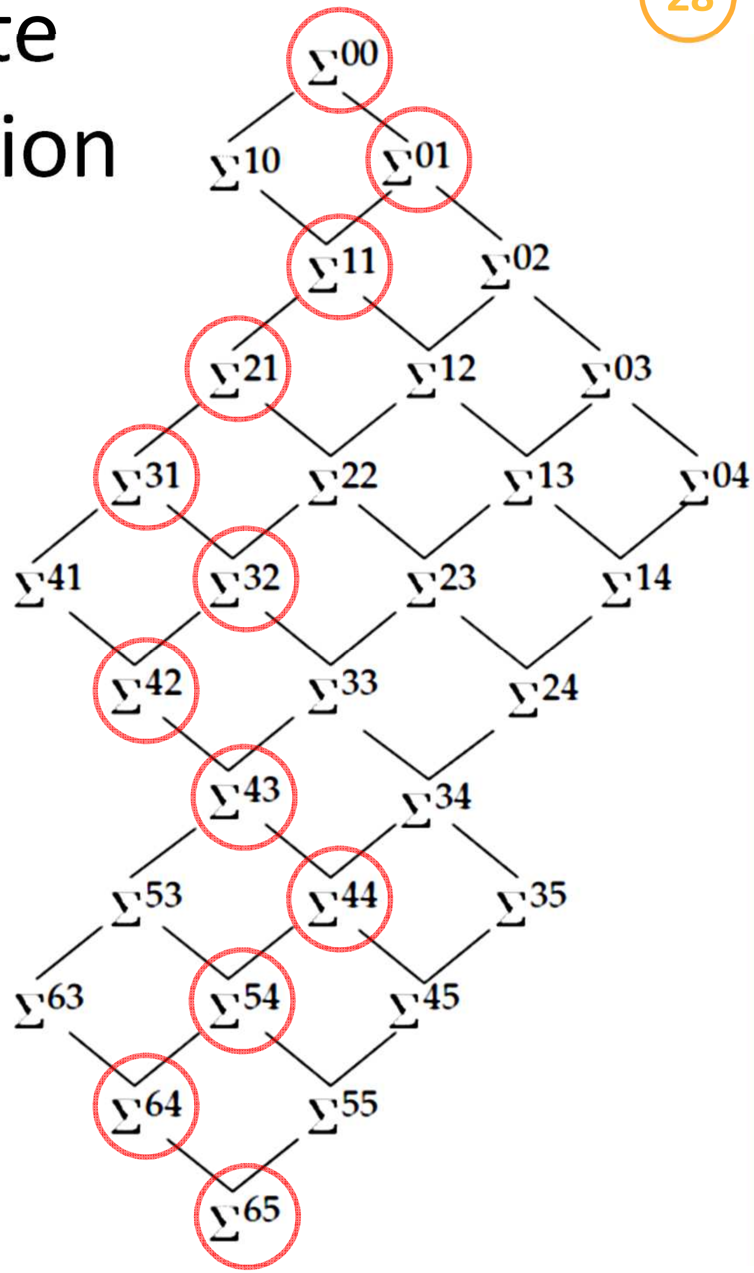
- A run $\mathbf{R = e^1\ e^2\ e^3\ \ldots}$ results in a sequence of global states $\Sigma^0\ \Sigma^1\ \Sigma^2\ \Sigma^3\ \ldots$, where $\boldsymbol{\Sigma^0 = (\sigma_1^0, \sigma_2^0, \ldots, \sigma_n^0)}$ is initial global state
- If run is consistent, the sequence of global states will be consistent as well
- Each consistent state Σ^i is obtained from previous state Σ^{i-1}, by executing a single event
- For two consecutive and consistent states in run R, we say that Σ^{i-1} leads-to Σ^i in R

Consistency of observed/constructed state

- Let $\rightharpoonup_R$ denote the transitive closure of the leads-to relation in a given run R
- We say that Σ' is reachable from Σ in run R, iff $\Sigma \rightharpoonup_R \Sigma'$
- The set of all consistent states of a computation, along with the leads-to relation, defines a lattice

Global state transformation





$e_2^1\ e_1^1\ e_1^2\ e_1^3\ e_2^2\ e_1^4\ e_2^3\ e_2^4\ e_1^5\ e_1^6\ e_2^5$

$\Sigma^{00}\ \Sigma^{01}\ \Sigma^{11}\ \Sigma^{21}\ \Sigma^{31}\ \Sigma^{32}$

$\Sigma^{42}\ \Sigma^{43}\ \Sigma^{44}\ \Sigma^{54}\ \Sigma^{64}\ \Sigma^{65}$

Today's lecture

- Going from concurrent systems (CS) to distributed systems (DS)
- What is meant by DS?
- Concepts: global state, cuts and runs
- Consistency of observed/constructed state
- **DS properties – fail safe, fault tolerant**
- Case study – mutual exclusion in DS
- Next time

Properties of DS

- There are lots of different properties we may be interested in wrt. Distributed systems, including but not limited to:
  - Absence of dead-lock or starvation
  - Effiency wrt. memory usage or number of messages that needs to be exchanged
  - Robustness in the presence of errors

Errors in DS

- A ***fail-safe*** system is one that, in the event of failure, responds in a way that will cause no harm, or at least a minimum of harm, to other systems or danger to personnel.
- ***Fault-tolerance*** is the property that enables a system) to continue operating properly in the event of the failure of (or one or more faults within) some of its components. If its operating quality decreases at all, the decrease is proportional to the severity of the failure, as compared to a naïvely-designed system in which even a small failure can cause total breakdown.

Today’s lecture

- Going from concurrent systems (CS) to distributed systems (DS)
- What is meant by DS?
- Concepts: global state, cuts and runs
- Consistency of observed/constructed state
- DS properties – fail safe, fault tolerant
- **Case study – mutual exclusion in DS**
- Next time

Mutual exclusion in DS

- Mutual exclusion is also important for DS
  - Mostly for the same reasons as for other CS
  - Use of shared resource (file, hardware, net, bandwidth etc.)
- But in DS we don't have nice synchronization features like semaphores etc. nor a shared clock
- So we have to do it otherwise – it is in fact a well-studied subject – using some kind of message exchange

Mutual exclusion in DS

- DS mutual exclusion algorithms generally fall in one of two categories:
  - ***Permission based*** – a process wishing to enter its critical region exchanges messages with the other processes to achieve permission to do so
  - ***Token based*** – the process that holds the token has permission to enter its critical region. A process wishing to enter its critical region without holding the token, exchanges messages with the other processes to get hold of the token

Mutual exclusion in DS

- Desirable properties of algorithms
  - Freedom from deadlock
  - Freedom from starvation
  - Fairness
  - Fault tolerance
- Metrics of algorithms
  - Message complexity (size, contents, number)
  - Synchronization delay
  - Response time
  - Memory consumption

Simple mutual exclusion algorithm

- Simple round-robin token passing
- The token is simply circulated continously between all processes
- A process that wants to enter the CR, waits until the token comes around, then enters the CR
- When a process exits its CR, or if it doesn't need to enter its CR, it sends the token on to the next process

Round-robin token mutual exclusion

- How does this algorithm fare wrt:
  - Freedom from deadlock
  - Freedom from starvation
  - Fairness
  - Fault tolerance
  - Message complexity (size, contents, number)
  - Synchronization delay
  - Response time
  - Memory consumption

Round-robin token mutual exclusion

```
Global_initialization:
- Create processes and connect them in a
  Circle (circular graph)
- Create 1 token and send it one process
```

Round-robin token mutual exclusion

```
Process_init:
  need_token = false

Process_Enter_CR()
  need_token = true
  wait until token is received

  /* critical region */

  need_token = false
  send tokenmsg to next process
```

Round-robin token mutual exclusion

```
Process_Handle_ReceiveTokenMsg()
  if (!need_token)
    send tokenmsg to next process
```

Today's lecture

- Going from concurrent systems (CS) to distributed systems (DS)
- What is meant by DS?
- Concepts: global state, cuts and runs
- Consistency of observed/constructed state
- DS properties – fail safe, fault tolerant
- Case study – mutual exclusion in DS
- **Next time**

Next time

- Exercise on monday
  - Implement (BACI) and model (Spin) round-robin token passing for mutual exclusion
- The Neilsen-Mizuno alg. for mutual exclusion
  - An example of a “real” DS algorithm
- Reading material
  - Article "A Dag-Based Algorithm for Distributed Mutual Exclusion" by Neilsen & Mizuno. Focus is on the first 4 pages only, section 1-3.