Consistent Detection of Global Predicates in Asynchronous Systems with Crash Faults

Felix Gärtner

Darmstadt University of Technology, Germany, felix@informatik.tu-darmstadt.de

joint work with

Sven Kloppenburg

Systeam Engineering, Darmstadt, Germany, sven@syseng.de

(appears at SRDS 2000)

Motivation

"We are looking for software which also works in very large distributed systems."

Overview

- Recap: observation in (fault-free) asynchronous systems (7 slides).
- Recap: modalities *possibly* and *definitely* (2 slides).
- Observation in asynchronous systems with crash faults (5 slides).
- Modalities *negotiably* and *discernibly* (4 slides).
- Base idea of detection algorithms (6 slides).

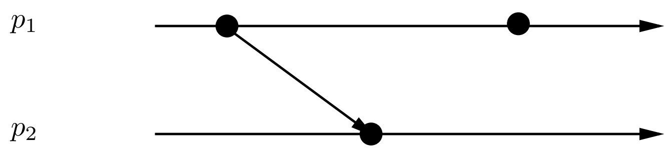
Asynchronous systems

- Set of n application processes $p_1, \ldots, p_n$ connected by a communication network.
- Communication by message passing using $send$ and $receive$ commands.
- Messages can take arbitrarily long.
- Processes can be arbitrarily slow.
- Very weak assumptions $\rightarrow$ very realistic model.

top

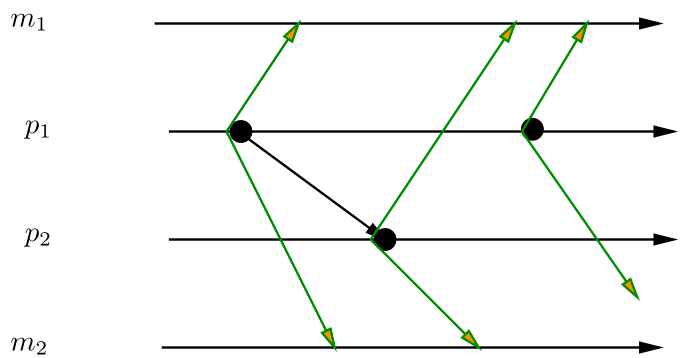
Observation in asynchronous systems

- Distributed computation in which events occur.
- For every relevant event, a *control message* is broadcast to a set of monitor processes .



Observation in asynchronous systems

- Distributed computation in which events occur.
- For every relevant event, a *control message* is broadcast to a set of monitor processes .



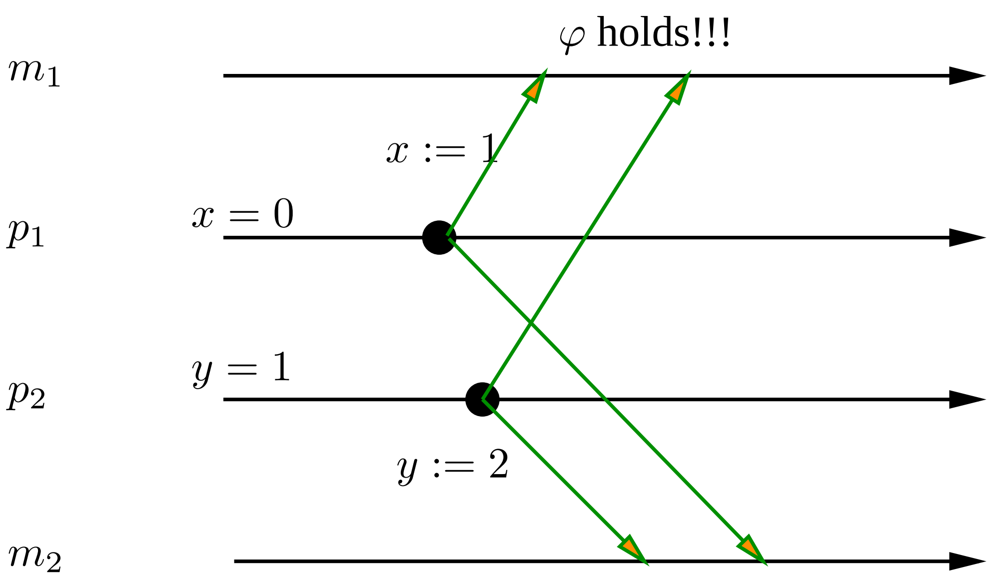
- Global state = "cut" through the diagram.
- Monitors construct a sequence of global states $\Sigma = S_1, S_2, \ldots$
- Σ is called an *observation*.

Predicate detection

- Given predicate φ on global states.
- Devise an algorithm with:
  - (safety) no false detections of φ.
  - (liveness) if φ holds, it is eventually detected.
- Naive approach: monitors check every S_i for φ.

Difficulties of observation

- Detection predicate is $\varphi \equiv x = y$
- "φ holds?" is not observer-invariant!



Lattice of all global states

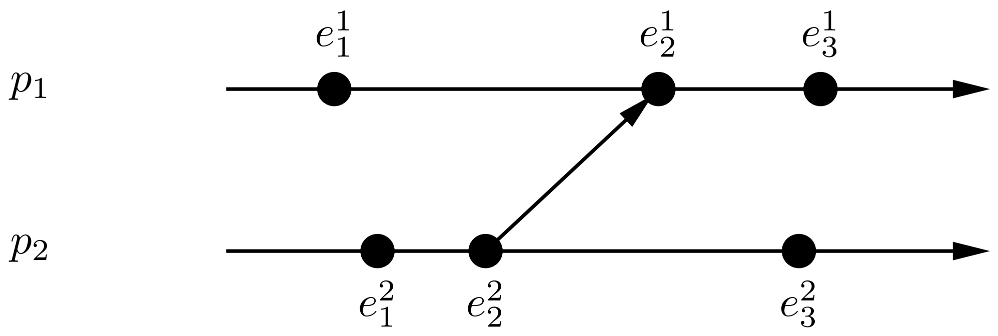
- Look at set of all (possibly) global states.
- Take relation on how one set evolves from another by executing a single event.

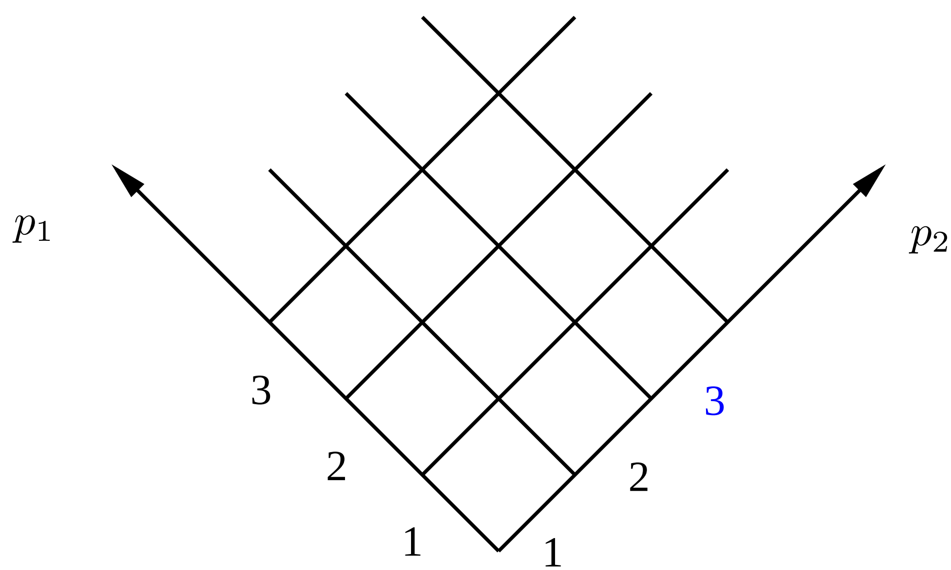
$\Rightarrow$ Lattice of global states.

- Observation is a path through the lattice.

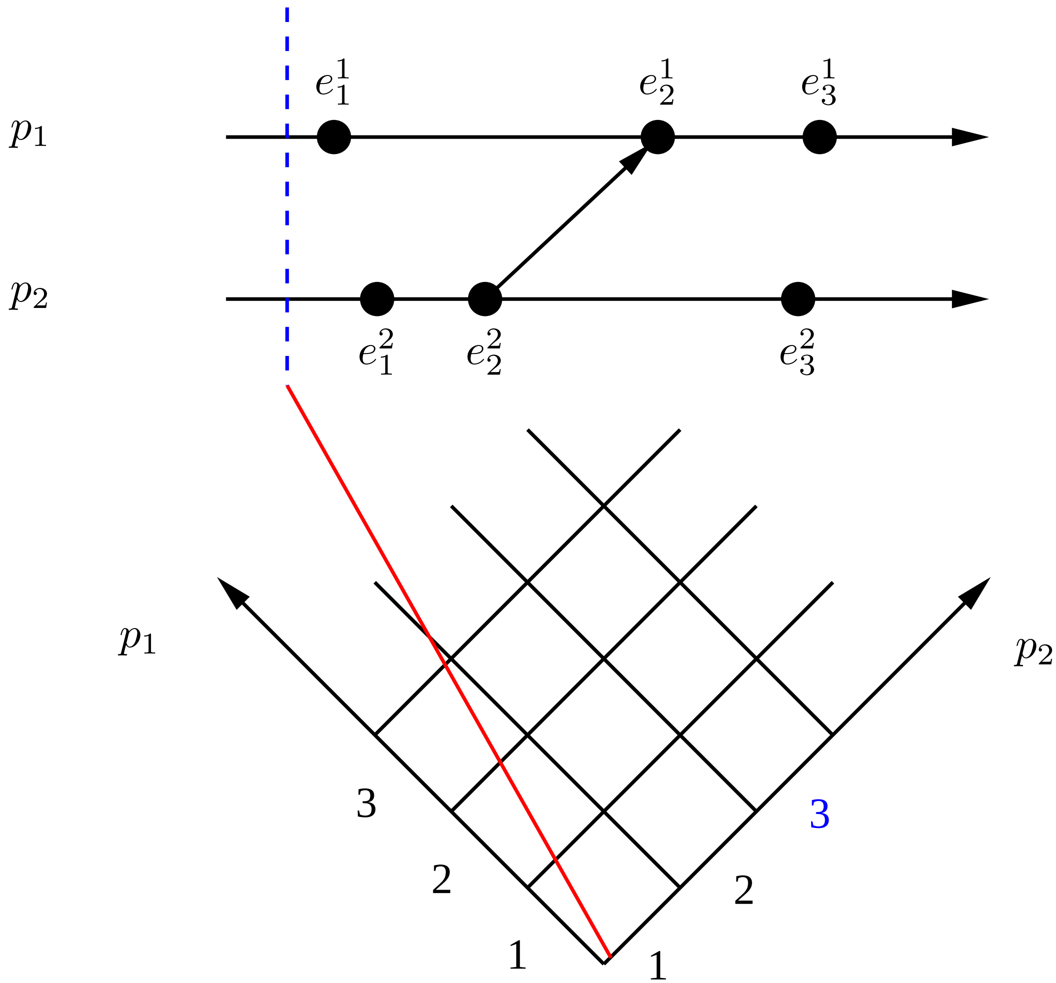
Lattice example

skip



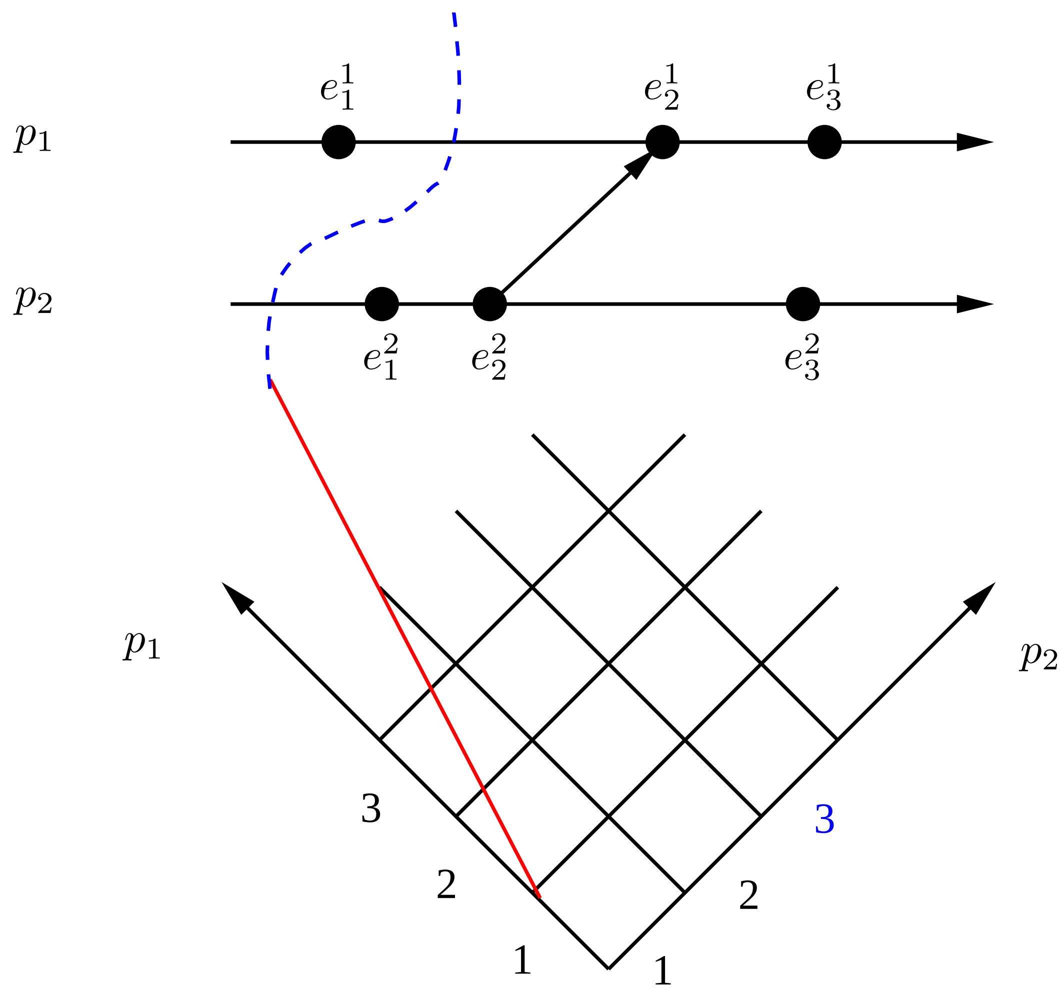


Lattice example



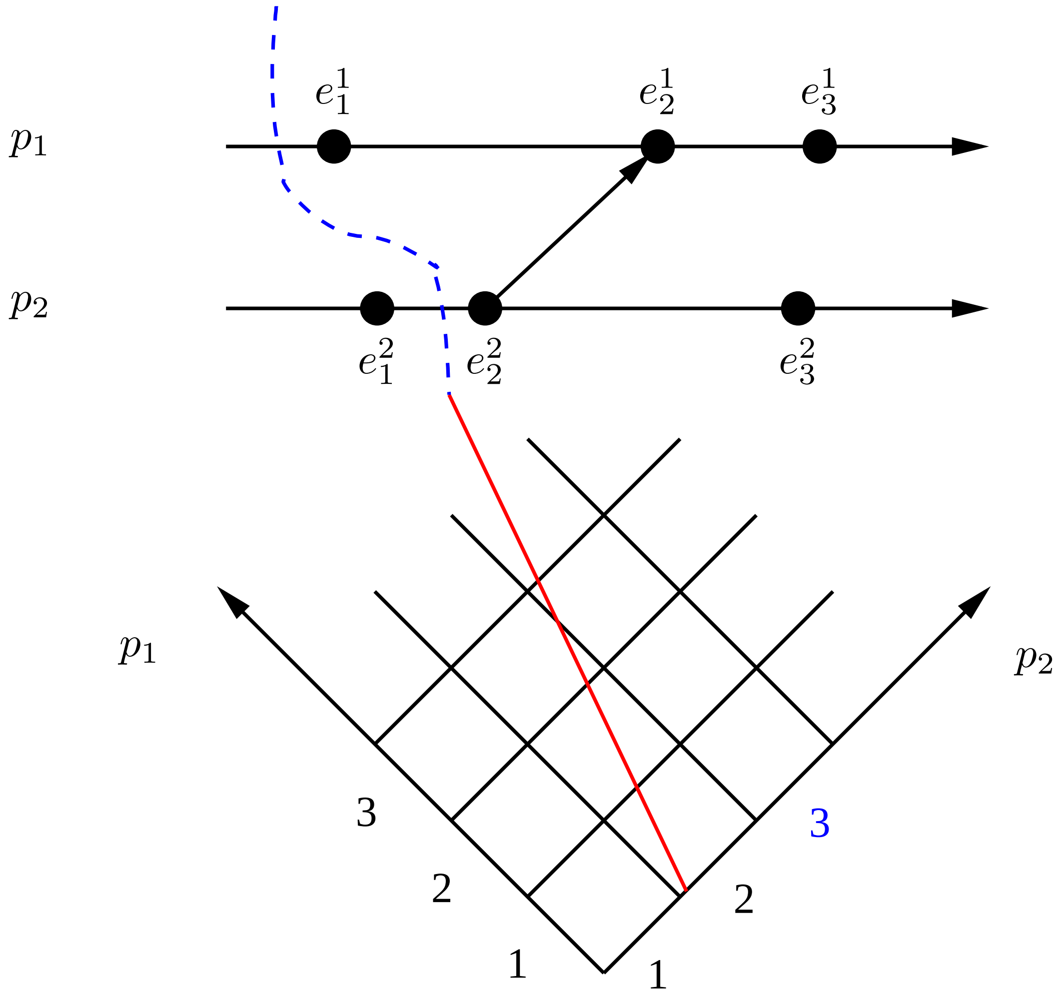
skip

Lattice example



skip

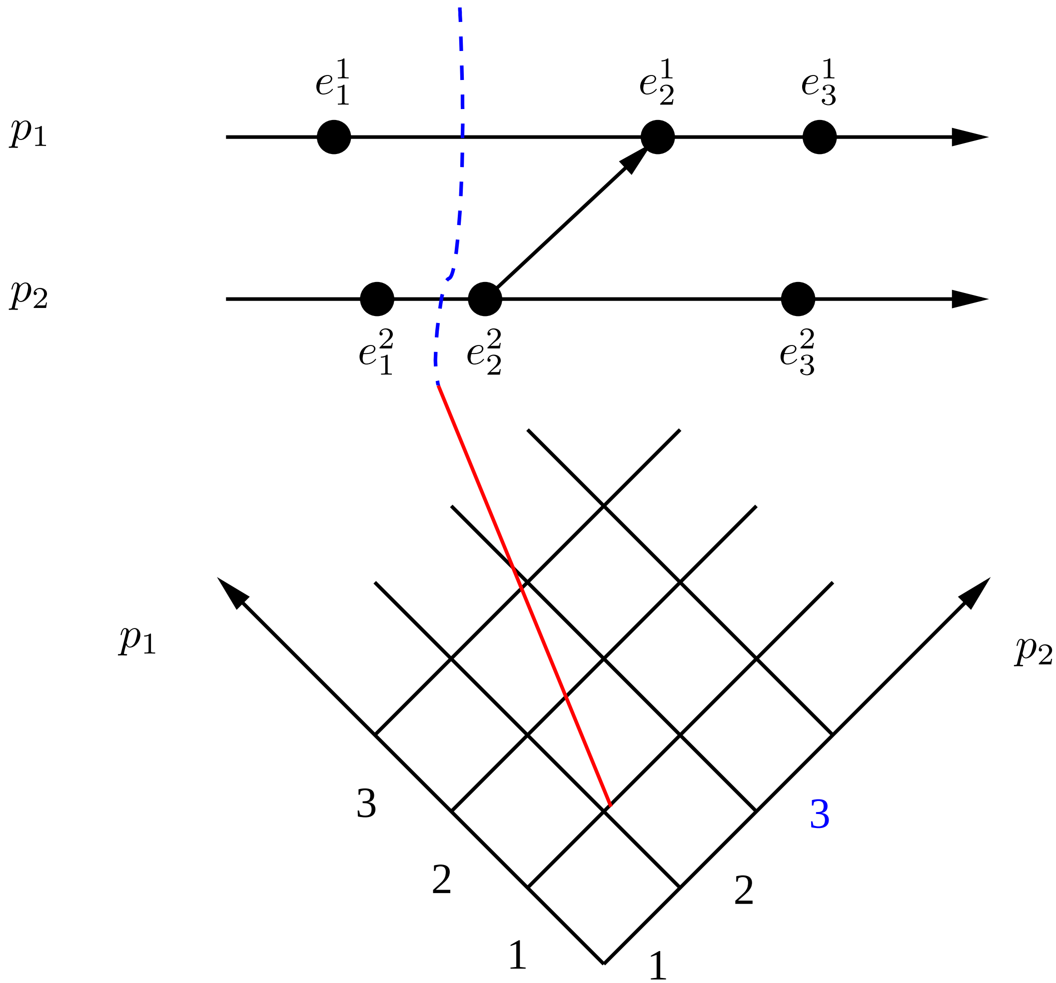
Lattice example



skip

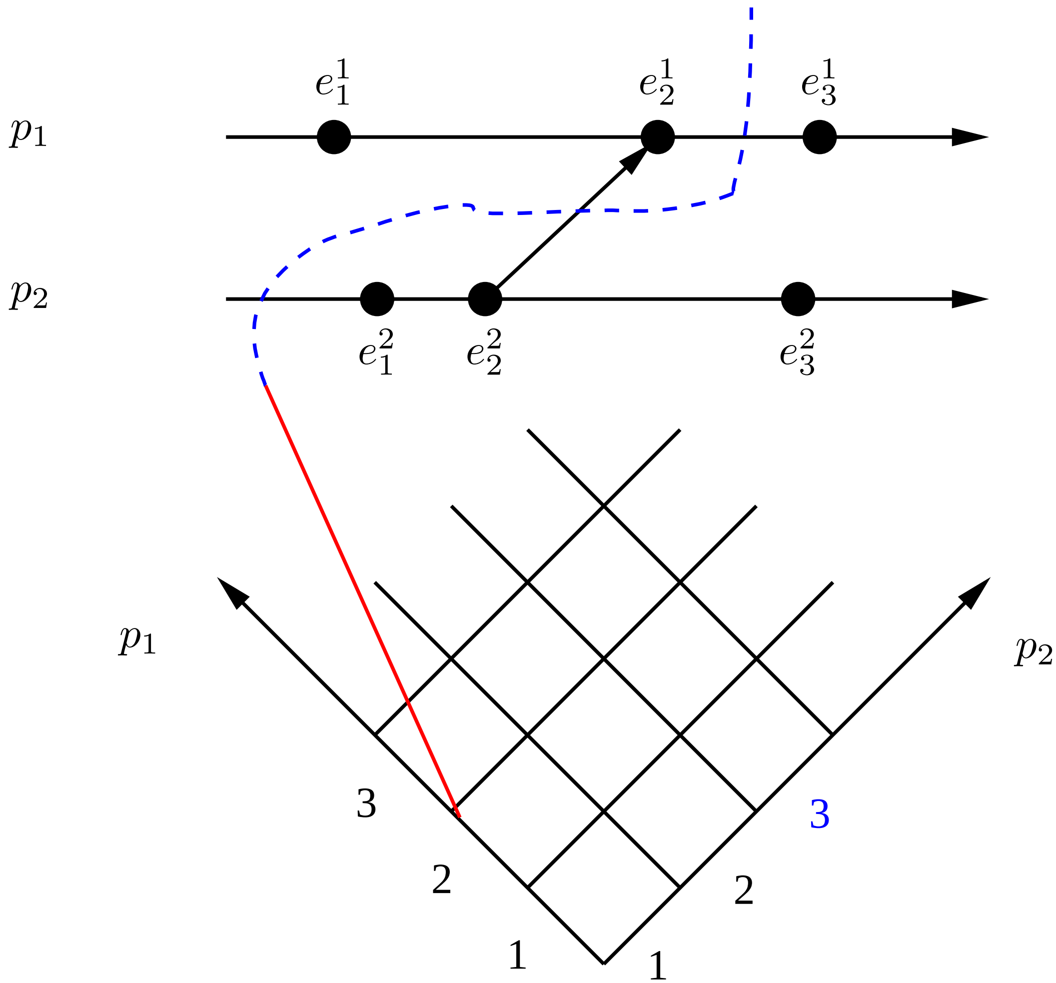
Lattice example

skip



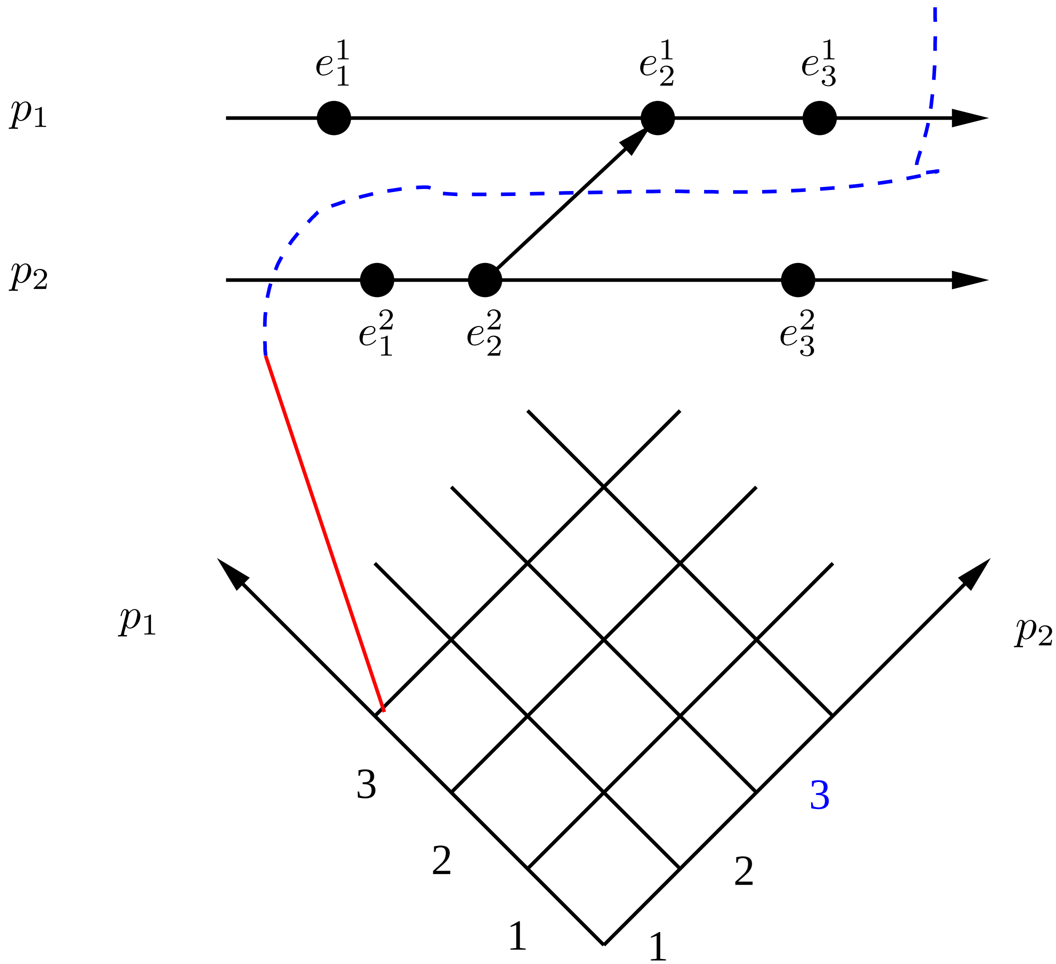
Lattice example

skip

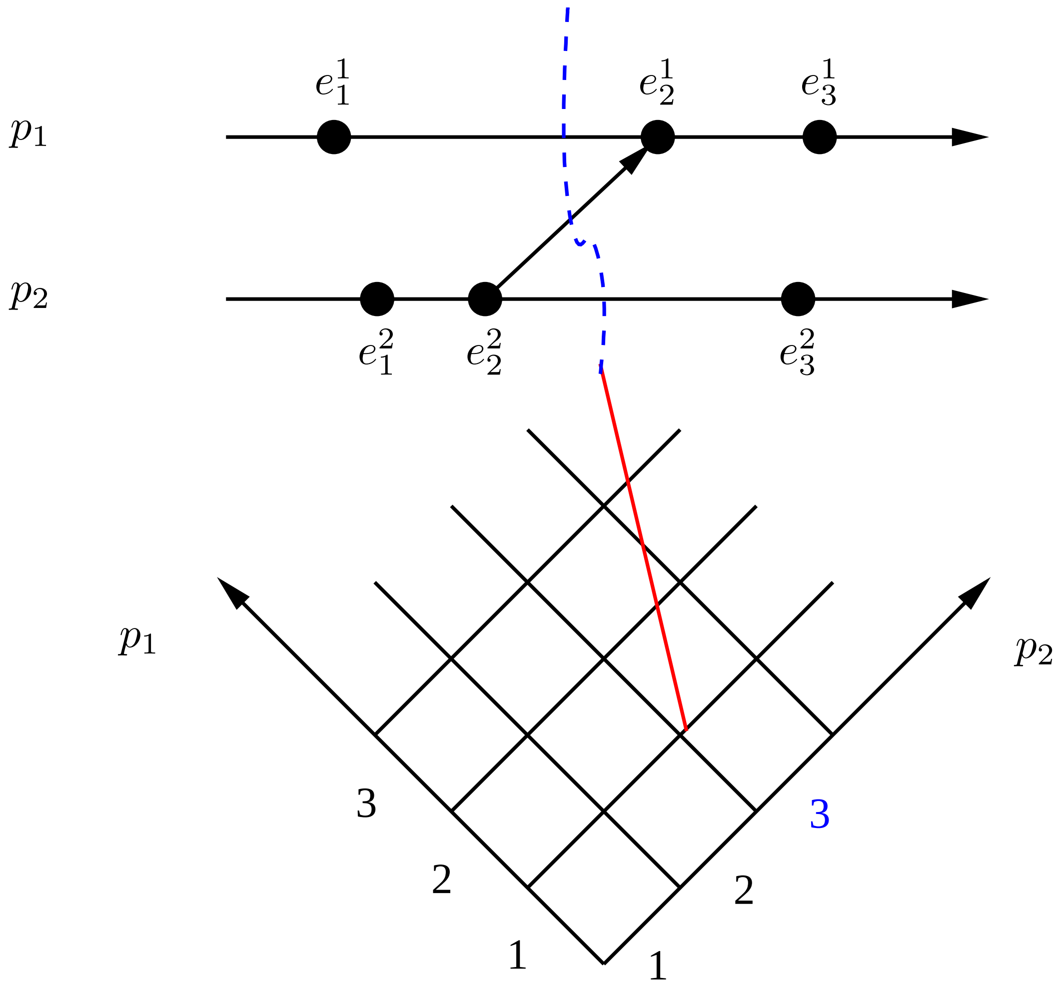


Lattice example

skip



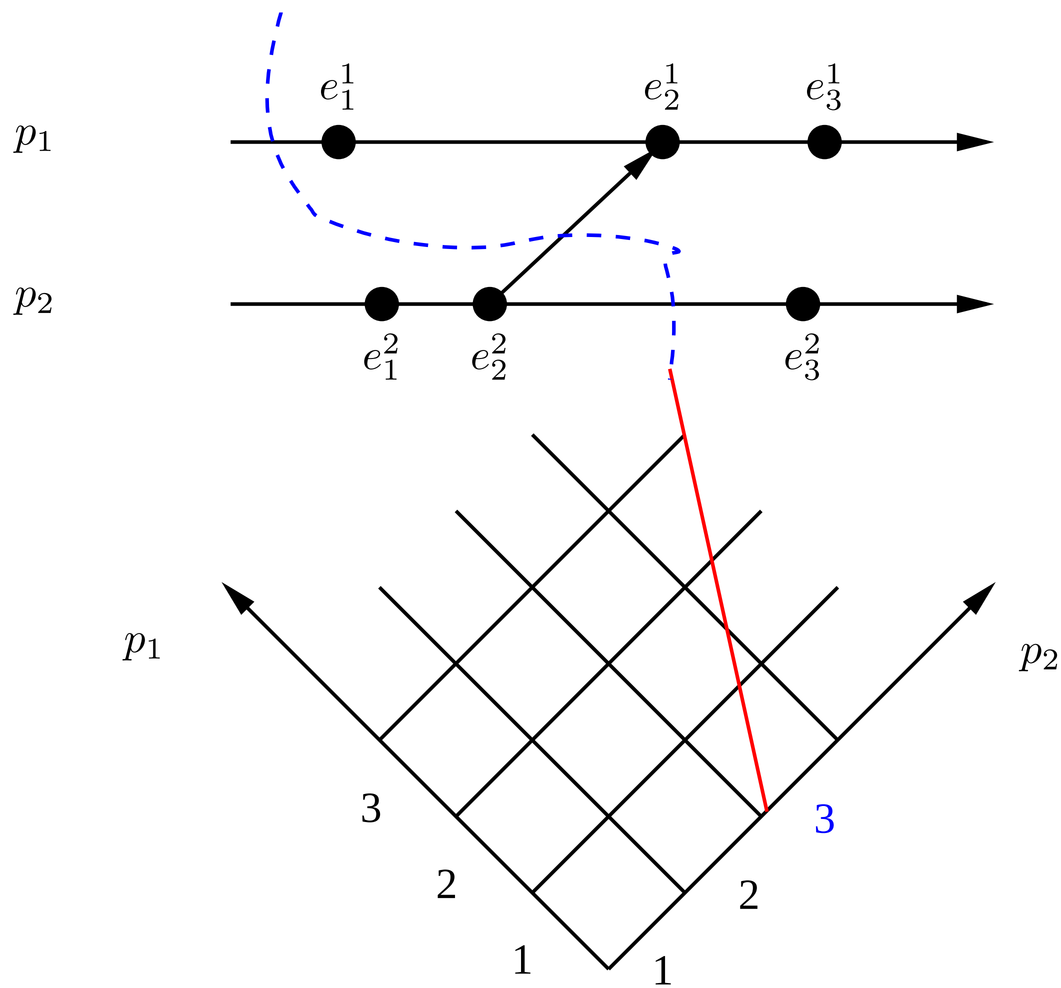
Lattice example



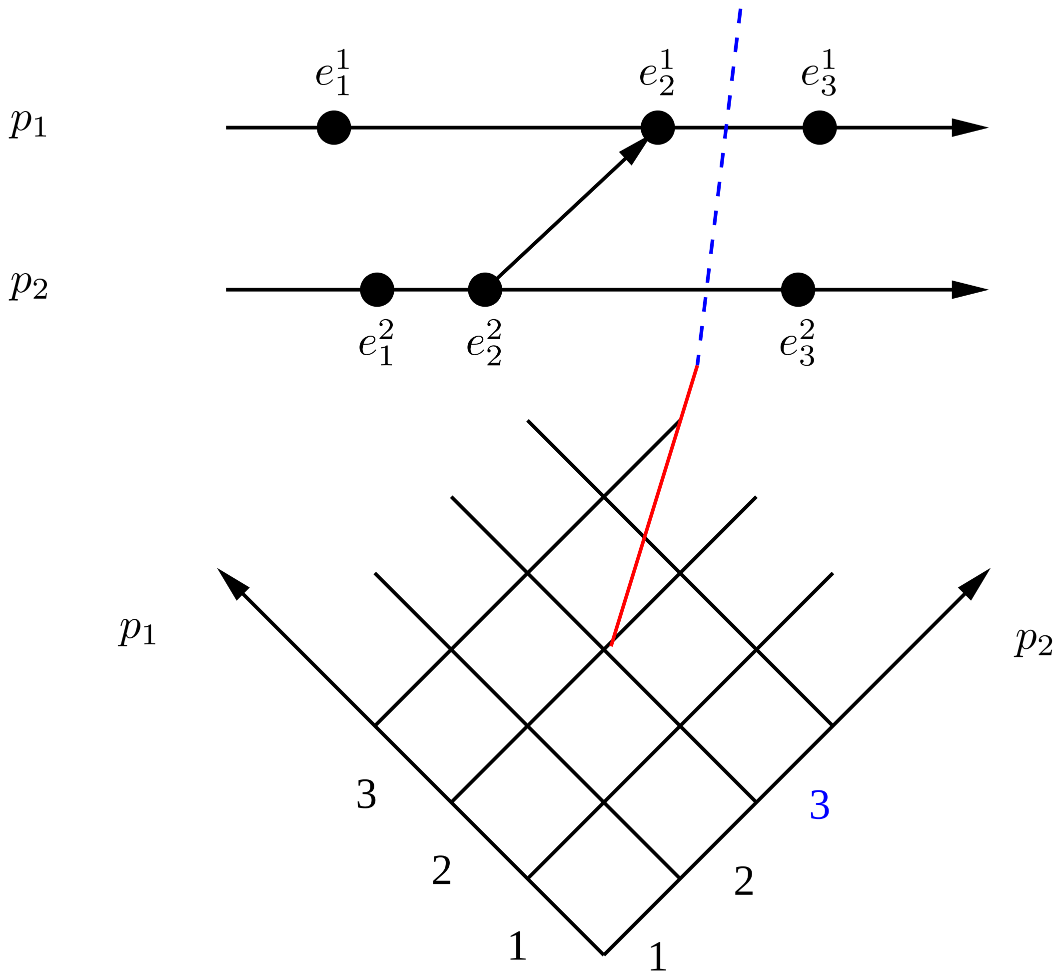
skip

Lattice example

skip



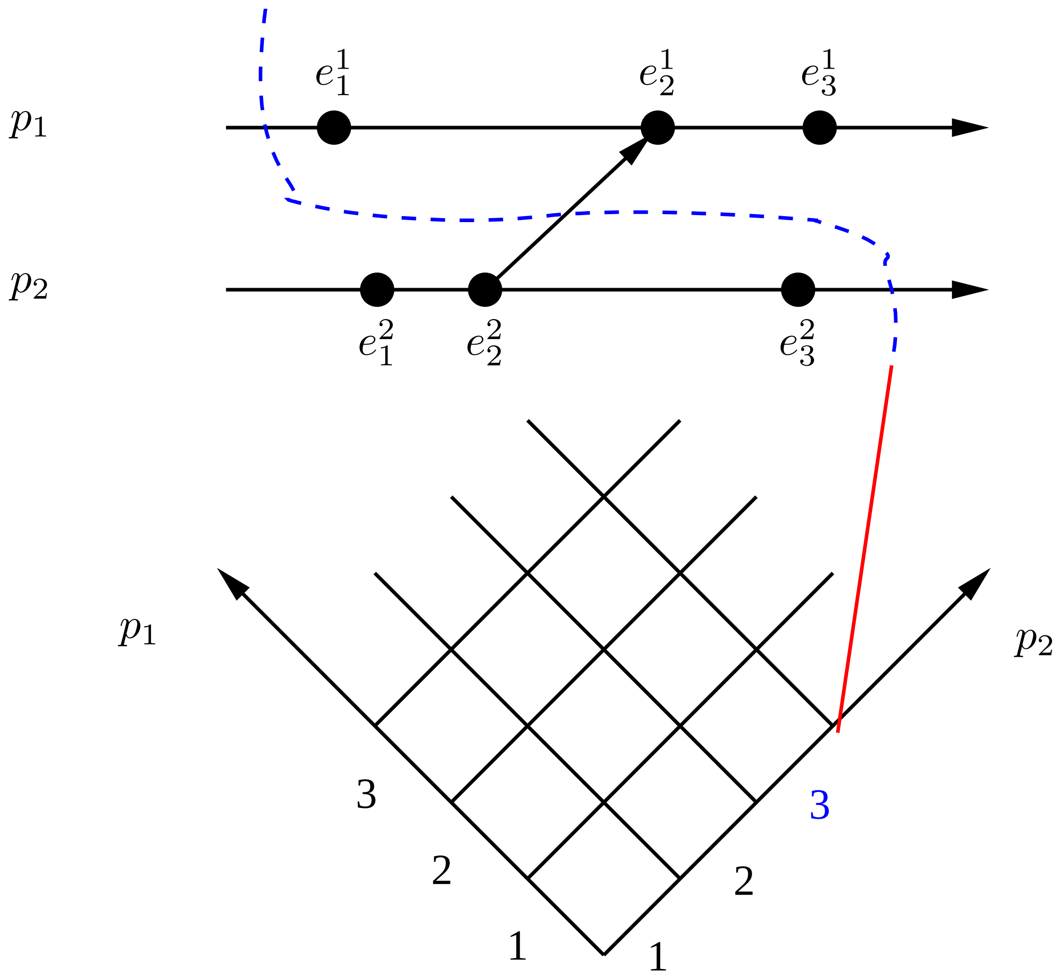
Lattice example



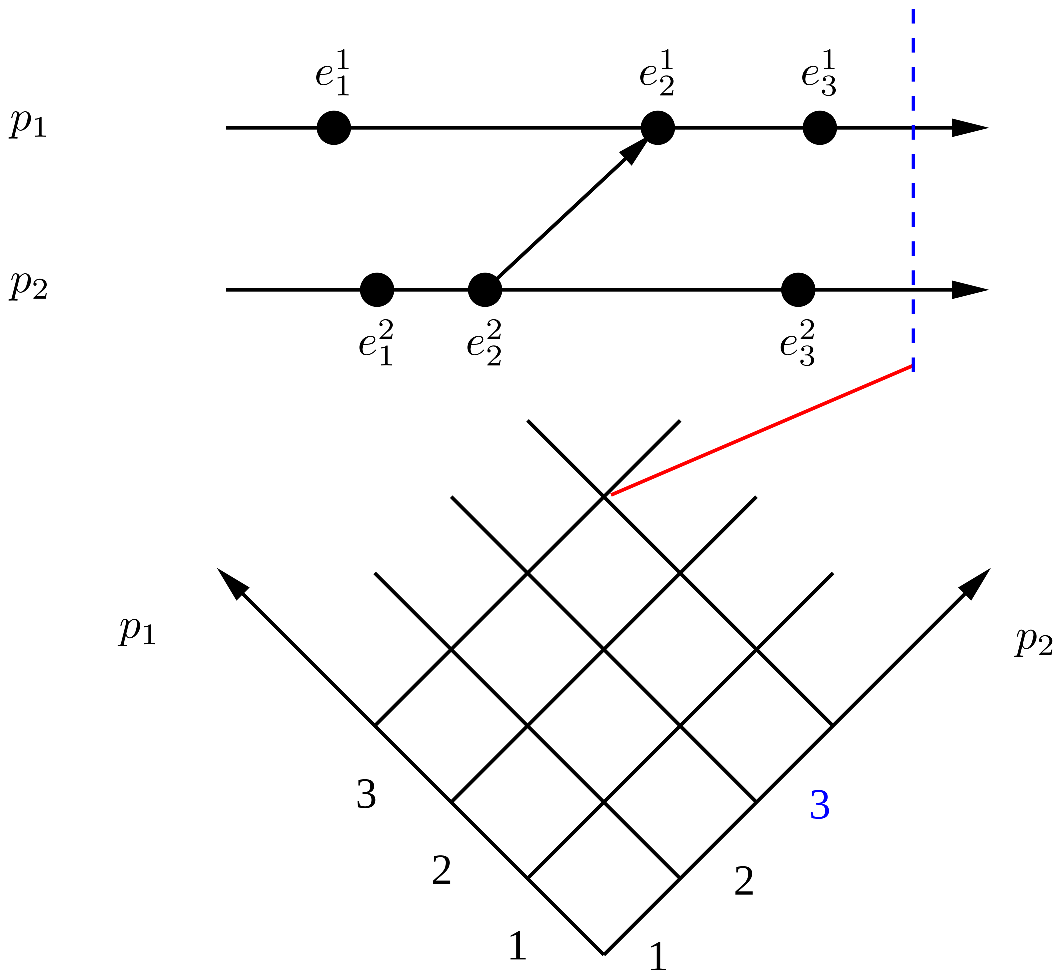
skip

Lattice example

skip



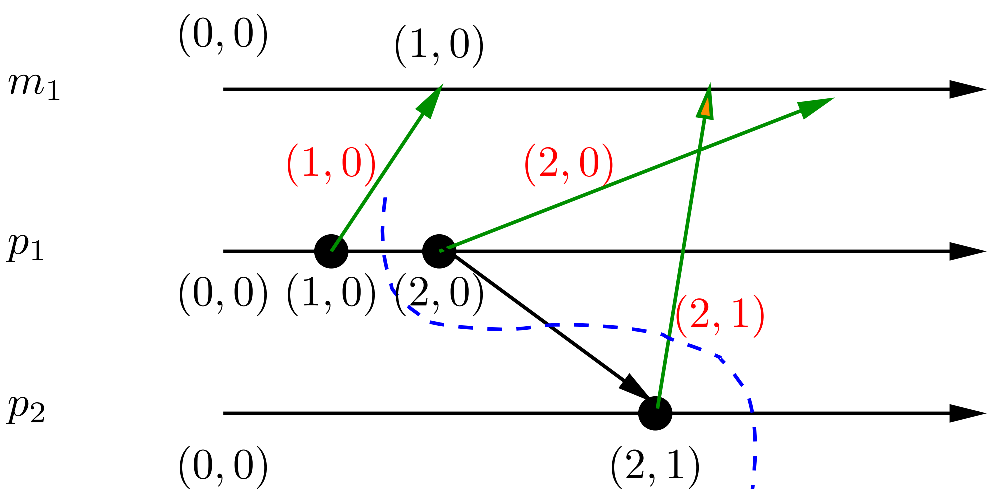
Lattice example



skip

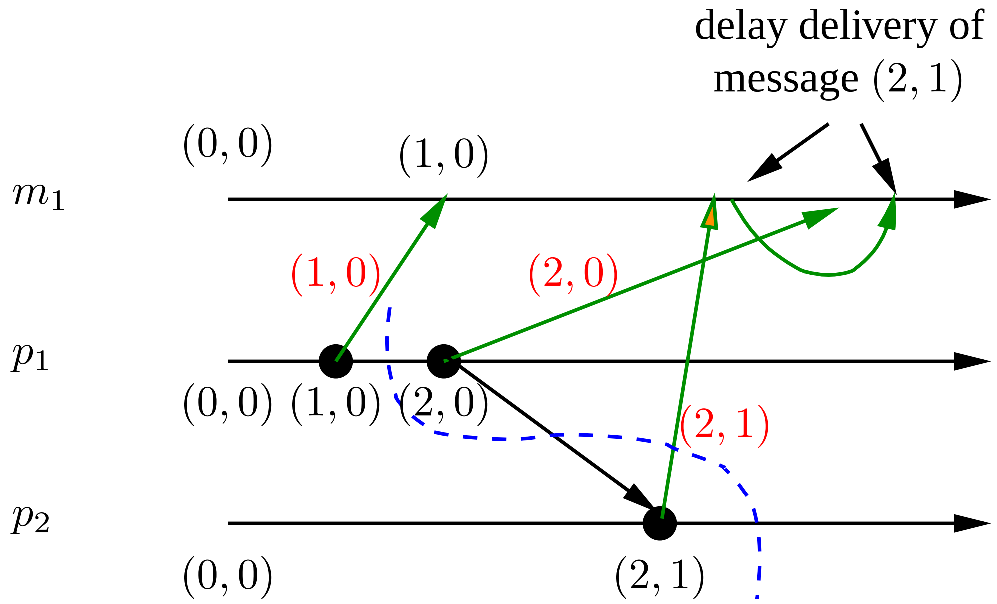
Consistent and inconsistent states

- Consistent state = respects causality
- Construct vector of local sequence numbers.
- Delay causally dependent control messages.



Consistent and inconsistent states

- Consistent state = respects causality
- Construct vector of local sequence numbers.
- Delay causally dependent control messages.



Modalities *possibly* and *definitely*

- Define observer-invariant notions:
  - $possibly(\varphi)$ holds iff there exists an observer which could see φ.
  - $definitely(\varphi)$ holds iff all observers at some time see φ.
- Observers construct and traverse state lattice to detect *possibly* or *definitely*.
- Safety requirement $\Box\varphi$: observers schould never detect $possibly(\neg\varphi)$.
- Liveness requirement $\Diamond\varphi$: detection of $definitely(\varphi)$ sufficient for validation.

Fault tolerance issues start here!

Faulty asynchronous systems

- Fault assumption = precise description about how and which components may fail.
- crash fault assumption = at most t processes simply stop executing steps.
- For the moment: restrict crash faults to application processes only (monitors always stay alive).
- Now study: predicate detection in asynchronous systems with crash faults.
- Only other work: Garg and Mitchell [3].

top

New types of predicates

- Predicate up_i refers to functional state of p_i.
- Can be used in predicates:
  - Process p_i crashed after 4th event:

$$\neg up_i \wedge ec_i = 4$$

  - Every process either commits or crashes:

$$\forall i : \neg up_i \vee commit_i = 1$$

  - Process p_i is waiting for a message from a crashed process:

$$j \in waiting_i \wedge \neg up_j$$

Failure detection

- Every monitor must manipulate up_i so that:
  - (safety) never $\neg up_i$ if p_i has not crashed.
  - (liveness) if p_i crashes, eventually truthify $\neg up_i$.
- This is impossible in asynchronous systems (FLP [2]).
- Terminology: failure detectors *suspect* and *rehabilitate* application processes.

Implementable failure detectors

- Can ensure liveness, but cannot avoid false suspicions.
- Best we can do: a non-crashing process is not permanently suspected [4].
- For observation purposes: add causality information to suspicions:
  - "m_j suspects p_i after event e_k on p_i."
  - "m_j rehabilitates p_i after event e_k on p_i."
- Assume: between two events at most one suspicion and rehabilitation.

Lattice over extended state space

- Treat up_i as a variable on p_i.
- Suspicion/rehabilitation is a simple state change of p_i.

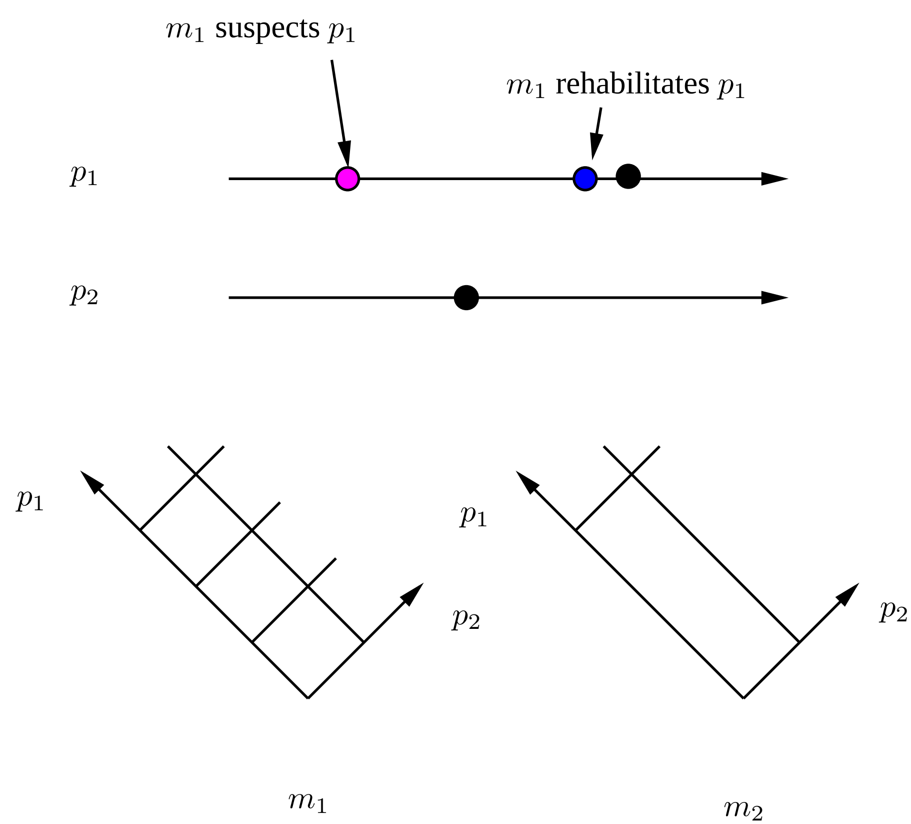
$\Rightarrow$ Extended state space.

- Change of up in consistent states yields again consistent states.

$\Rightarrow$ Integration of suspicions/rehabilitations into state lattice yields new lattice (over extended state space).

Per monitor lattice

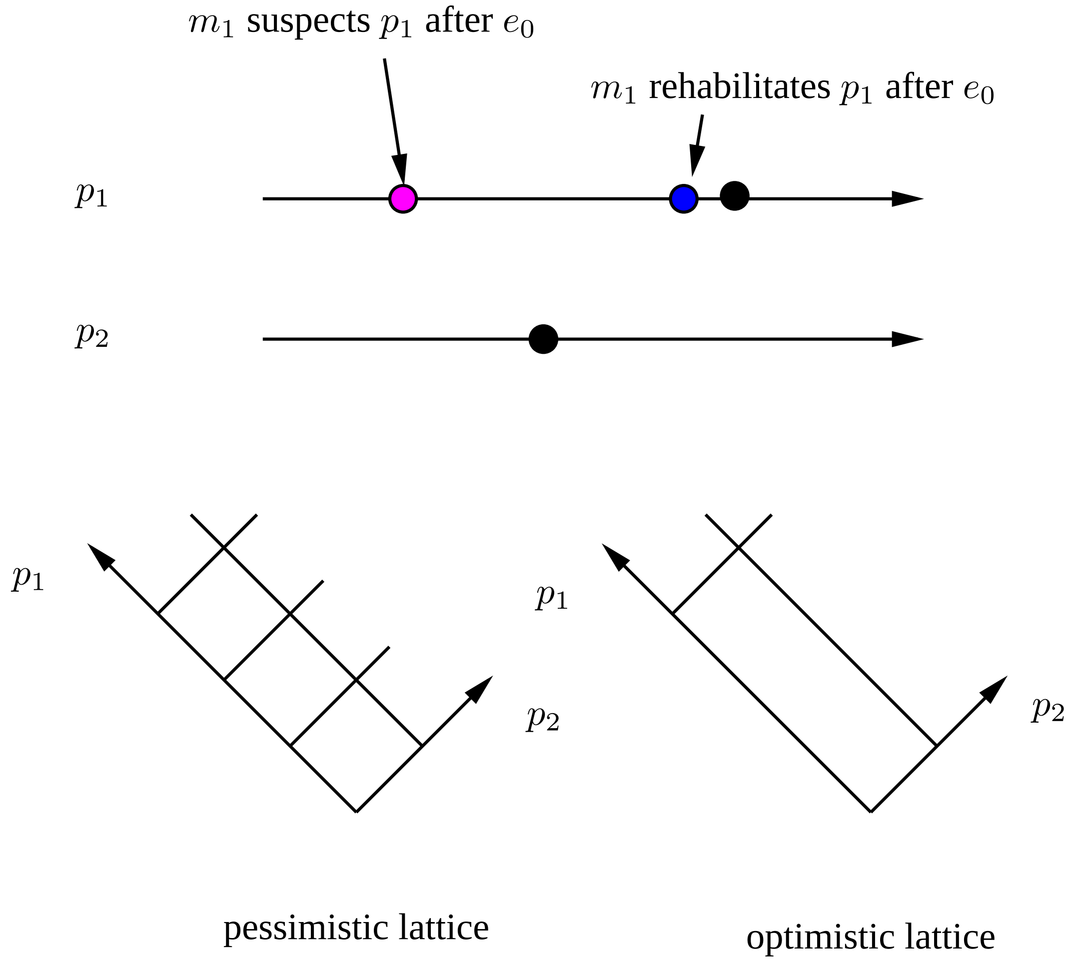
- Due to false suspicions monitors construct different state lattices.
- *possibly/definitely* not observer-invariant.



Global failure detector semantics

- Problem: false suspicions.
- Solution: define "global" failure detector semantics.
- p_i is suspected after e_k iff . . .
  - (pessimistic) $\exists$ a monitor which suspects p_i after e_k.
  - (optimistic) $\forall$ monitors suspect p_i after e_k.
- Can define pessimistic and optimistic state lattice.

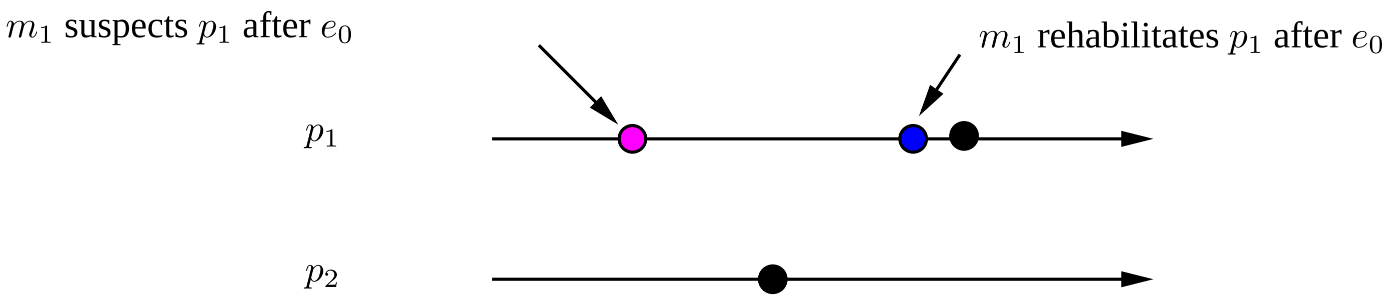
Optimistic/pessimistic state lattice example

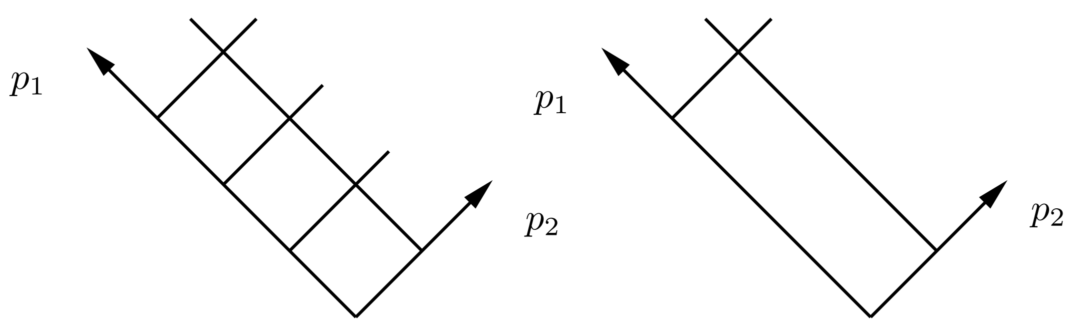


New modalities

top

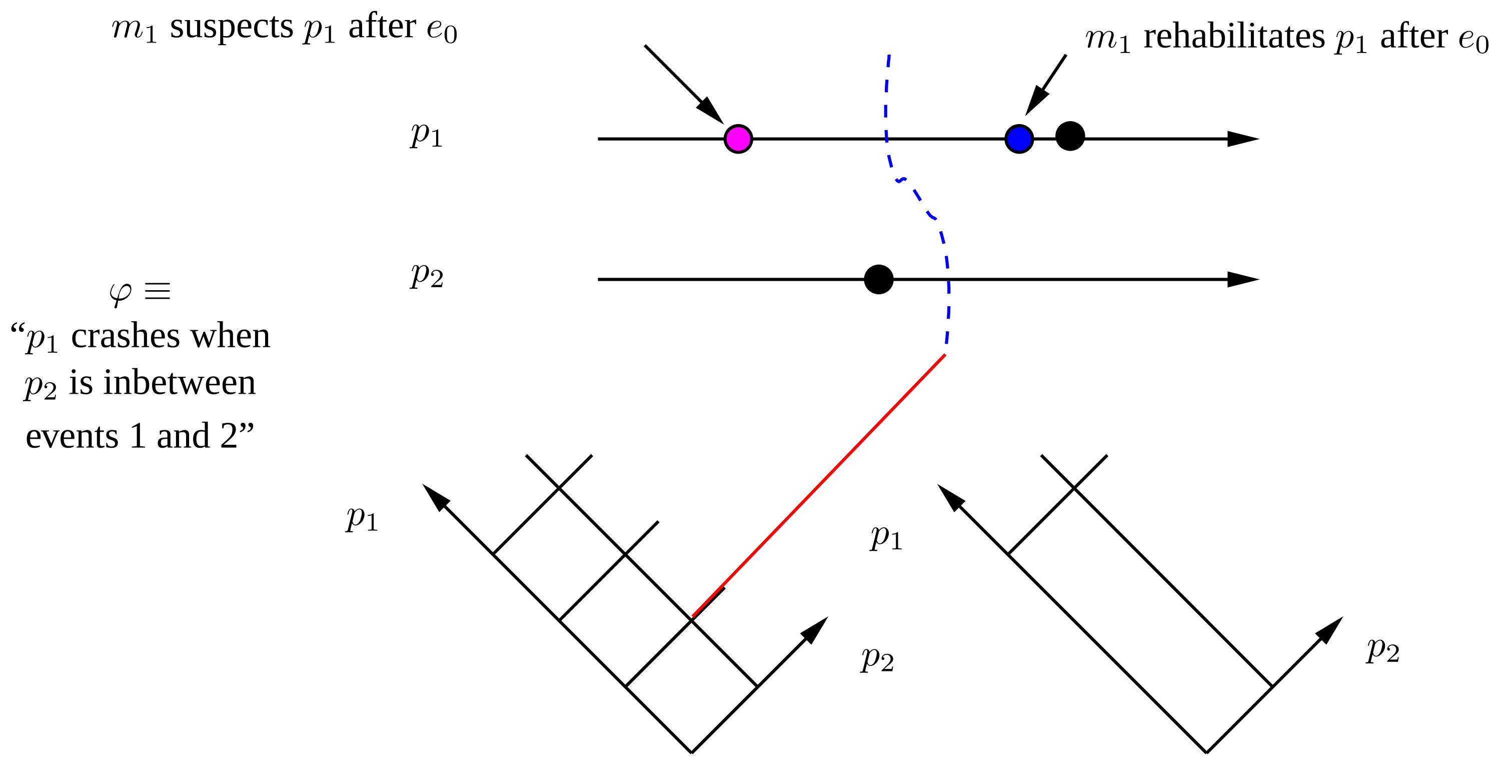
- Given predicate φ on extended state space.
- $negotiably(\varphi)$ holds iff $possibly(\varphi)$ holds on pessimistic state lattice.
- $discernibly(\varphi)$ holds iff $definitely(\varphi)$ holds on optimistic state lattice.





New modalities

- Given predicate φ on extended state space. top
- $negotiably(\varphi)$ holds iff $possibly(\varphi)$ holds on pessimistic state lattice.
- $discernibly(\varphi)$ holds iff $definitely(\varphi)$ holds on optimistic state lattice.

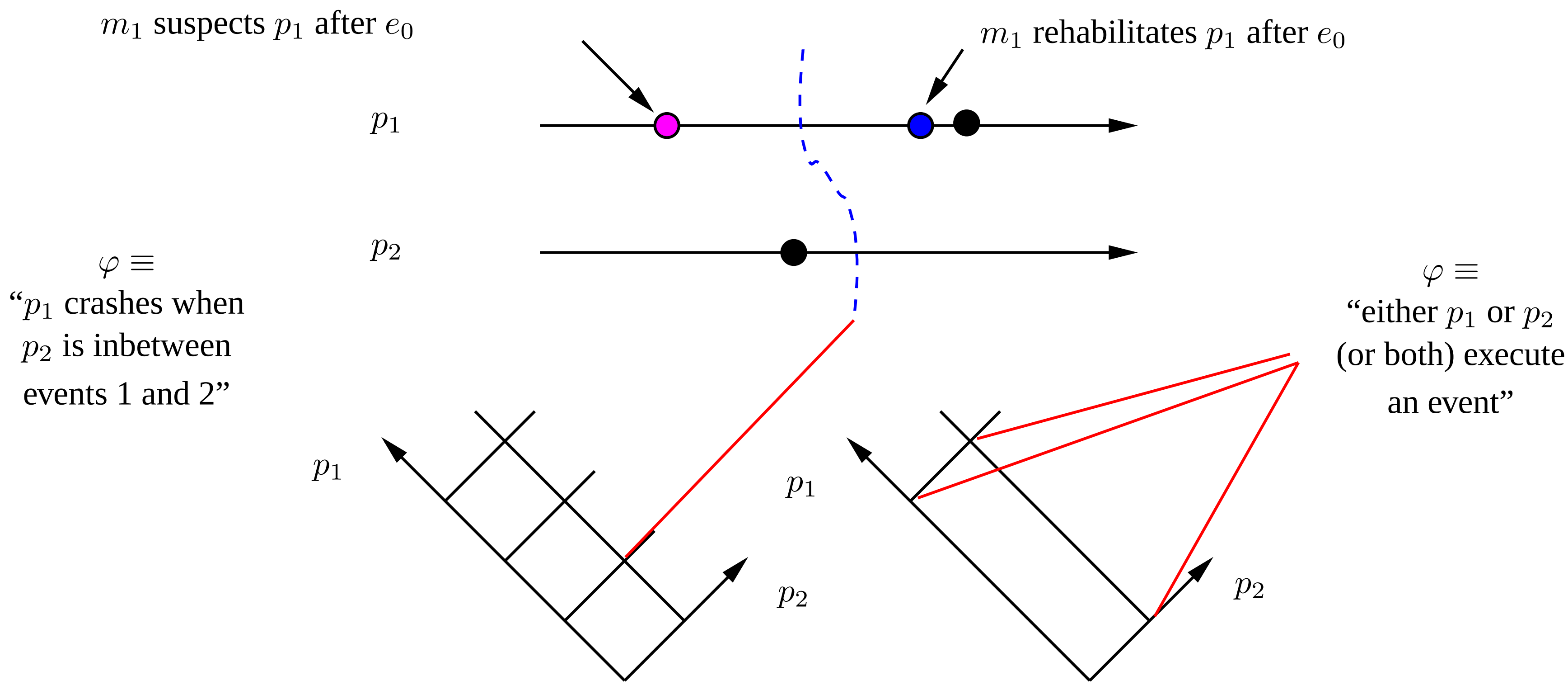


New modalities

- Given predicate φ on extended state space.

top

- $negotiably(\varphi)$ holds iff $possibly(\varphi)$ holds on pessimistic state lattice.
- $discernibly(\varphi)$ holds iff $definitely(\varphi)$ holds on optimistic state lattice.



Intuition behind new modalities

- Intuition of optimistic/pessimistic network protocols:
  - pessimistic: be careful all the time, take immediate action.

  $\Rightarrow$ use *negotiably* to trigger action.
  - optimistic: go ahead and hope for the best.

  $\Rightarrow$ use *discernibly* to ignore spurious suspicions.

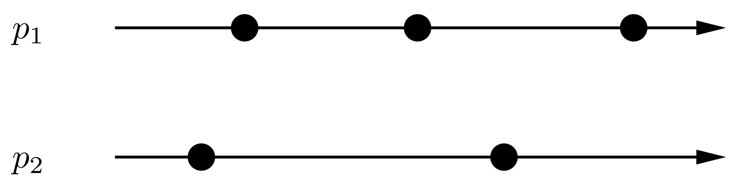
- Understandable in analogy to *possibly*/*definitely*:
  - Safety requirement $\Box\varphi$: take action if $negotiably(\neg\varphi)$ is detected.
  - Liveness requirement $\Diamond\varphi$: validated if $discernibly(\varphi)$ is detected.

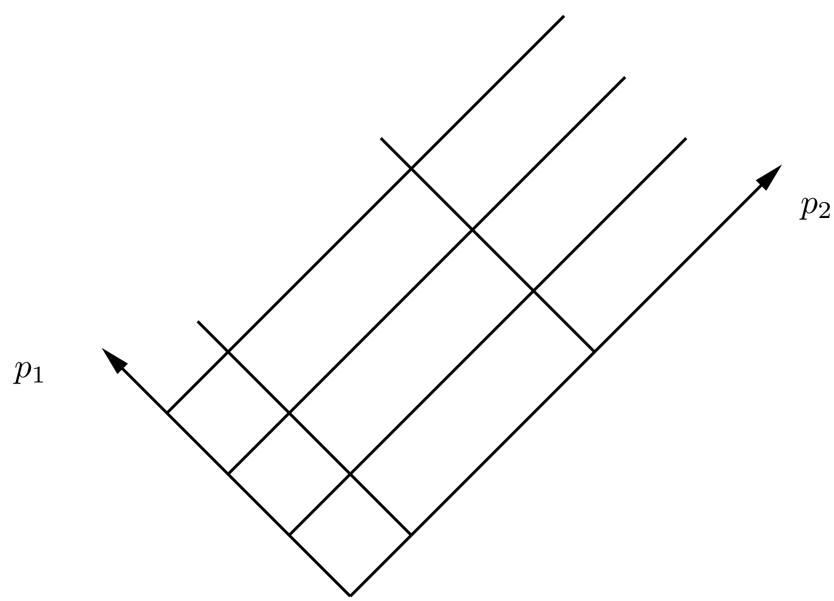
Detection algorithms

- Let monitors broadcast their suspicions to all other monitors.
- Eventually all monitor lattices converge.
- Can then do $possibly/definitely$ detection in observer invariant state lattices (use standard algorithms).

Lattice convergence example

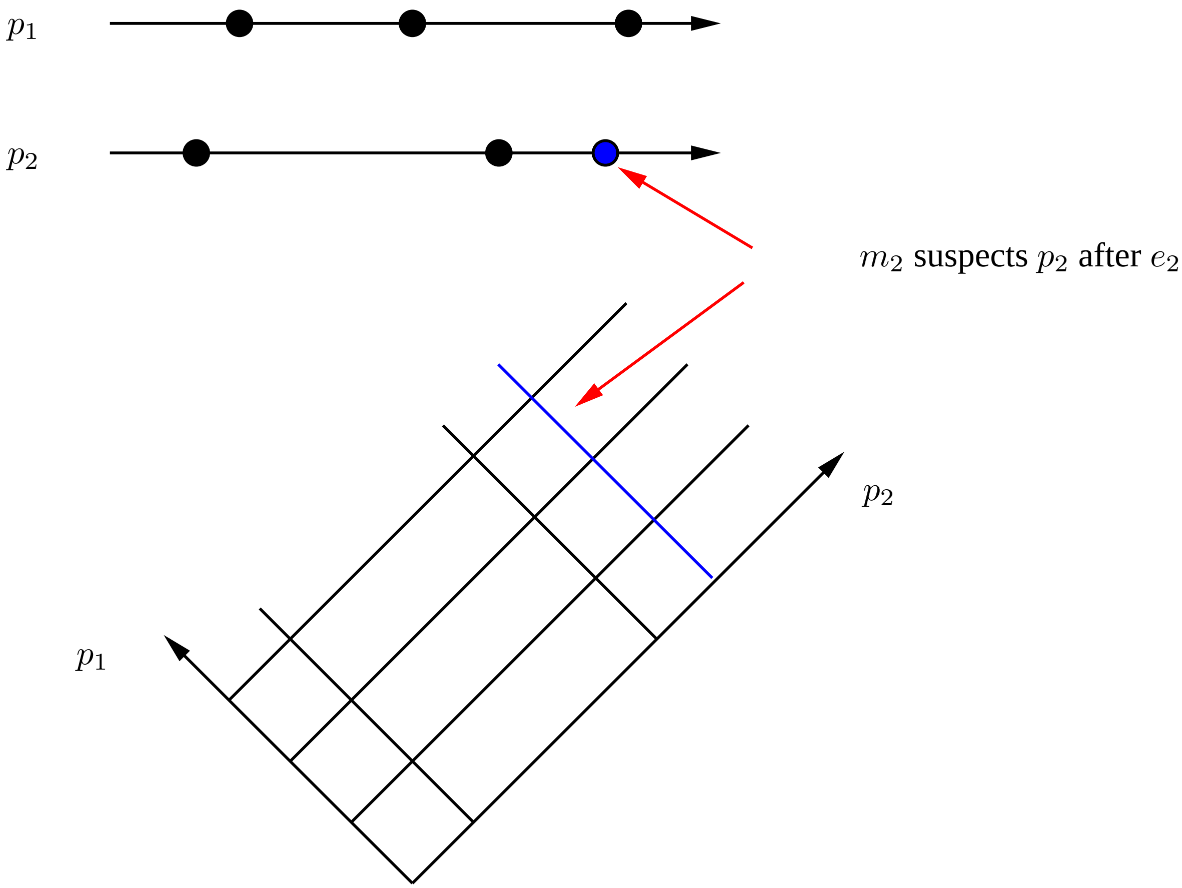
- No use starting detection on “unfinished” lattice!





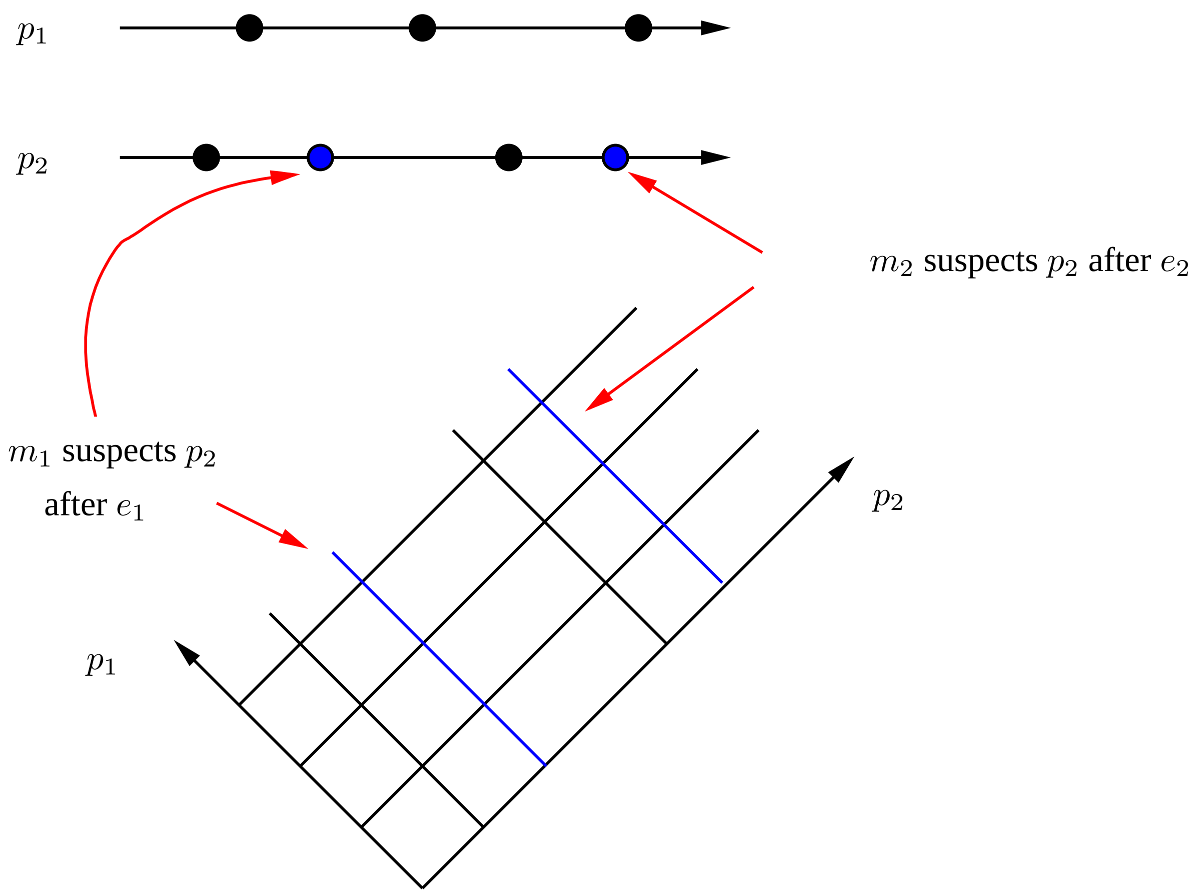
Lattice convergence example

- No use starting detection on “unfinished” lattice!



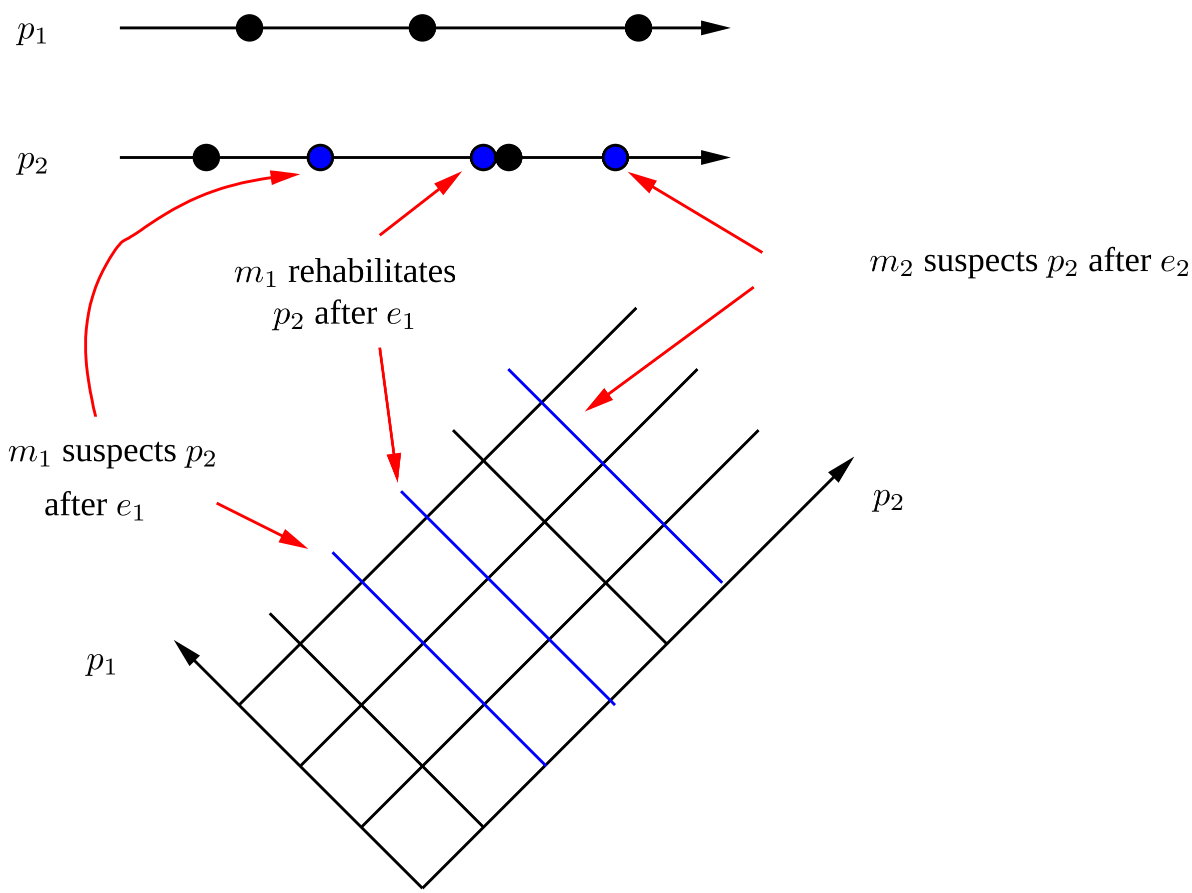
Lattice convergence example

- No use starting detection on “unfinished” lattice!



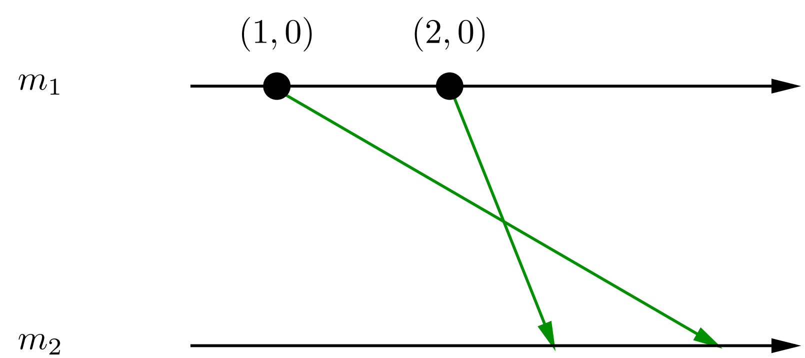
Lattice convergence example

- No use starting detection on “unfinished” lattice!



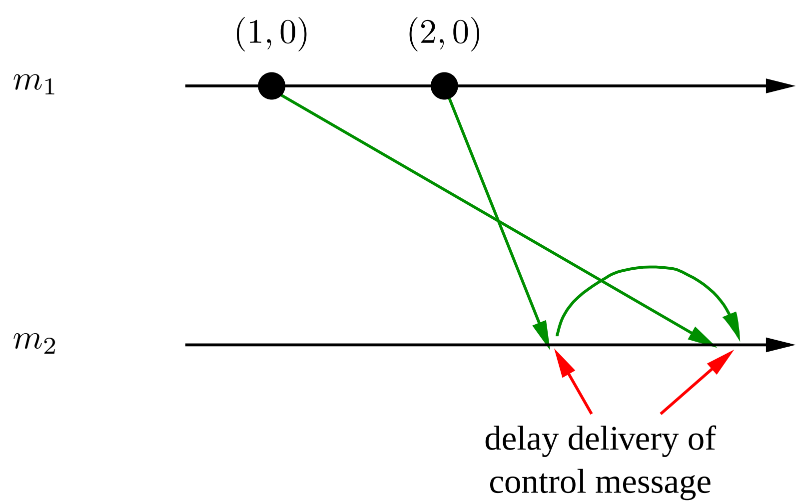
Causal broadcast of failure detector info

- Idea: causally broadcast failure detector events.
- You'll always get the "next one".



Causal broadcast of failure detector info

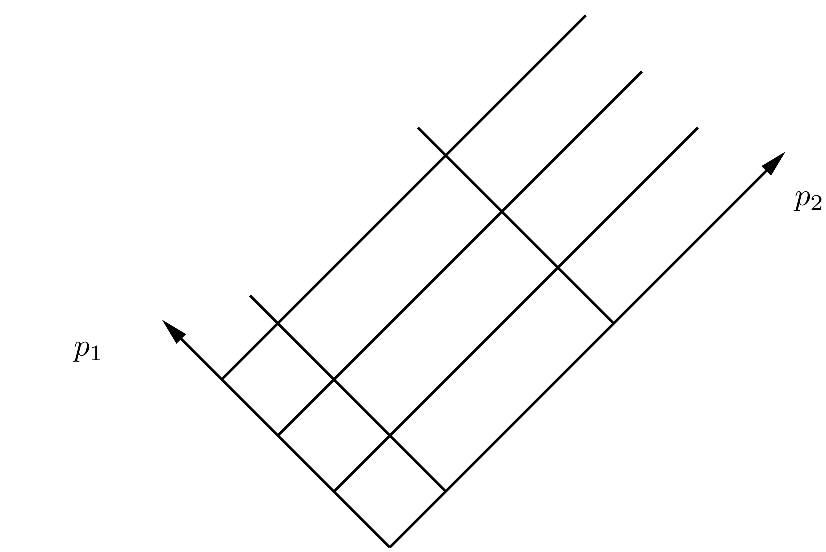
- Idea: causally broadcast failure detector events.
- You'll always get the "next one".



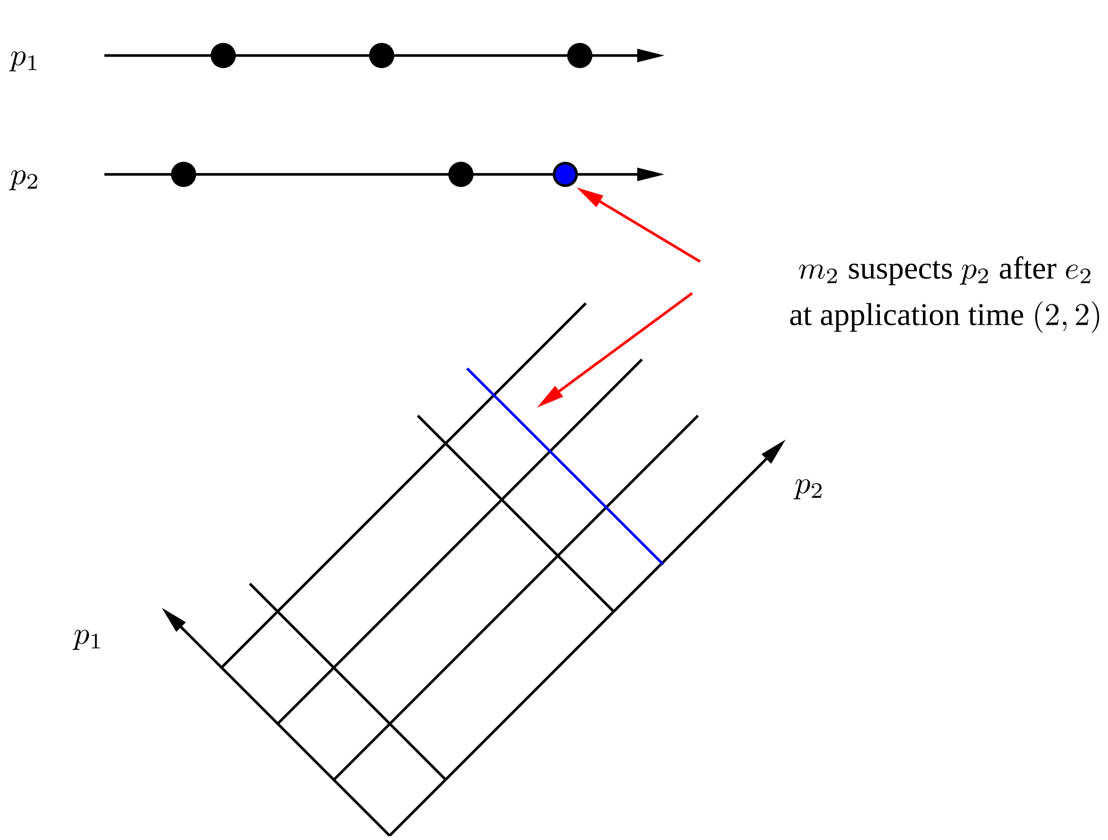
Settled region

- Causal broadcast of failure detector messages is useful!
- Monitors piggy back coordinates of most recent global state they have seen: per monitor stable region.
- Take intersection of all monitor regions: globally stable region.
- Steadily expand stable region, extract optimistic/pessimistic data and do $possibly/definitely$ detection on it.

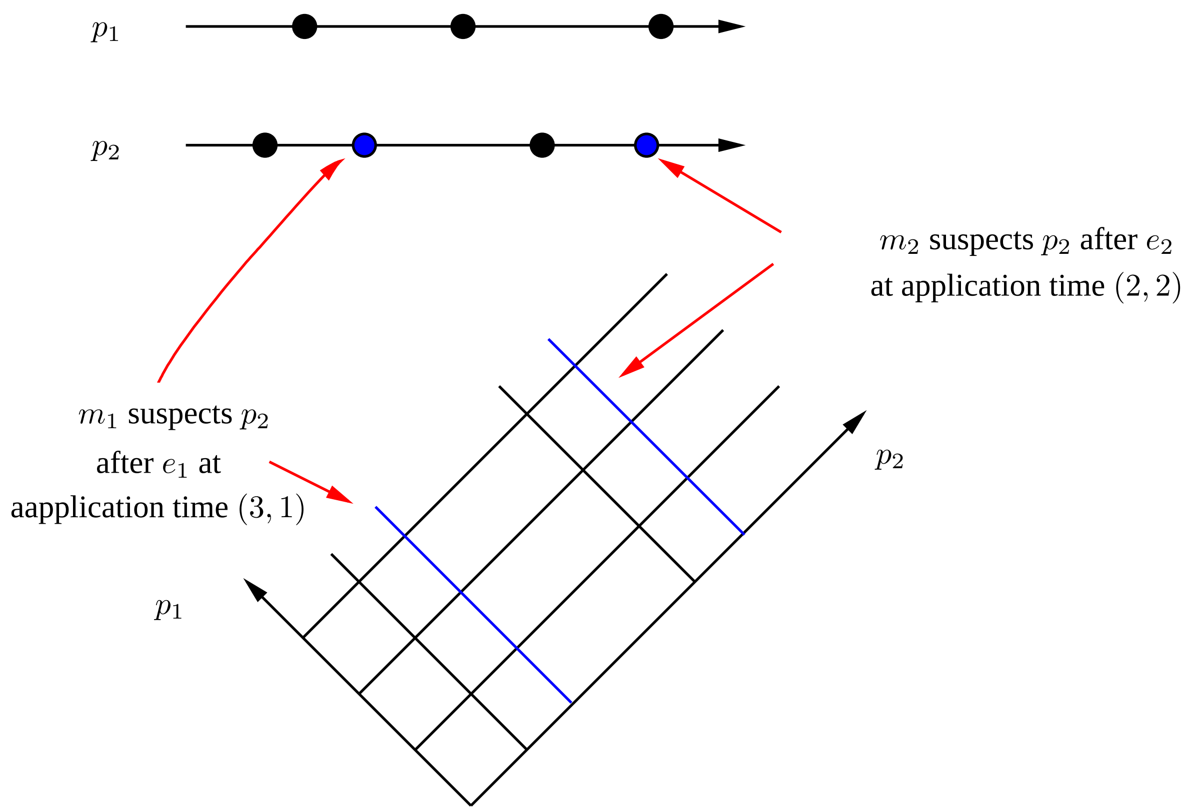
Settled region example



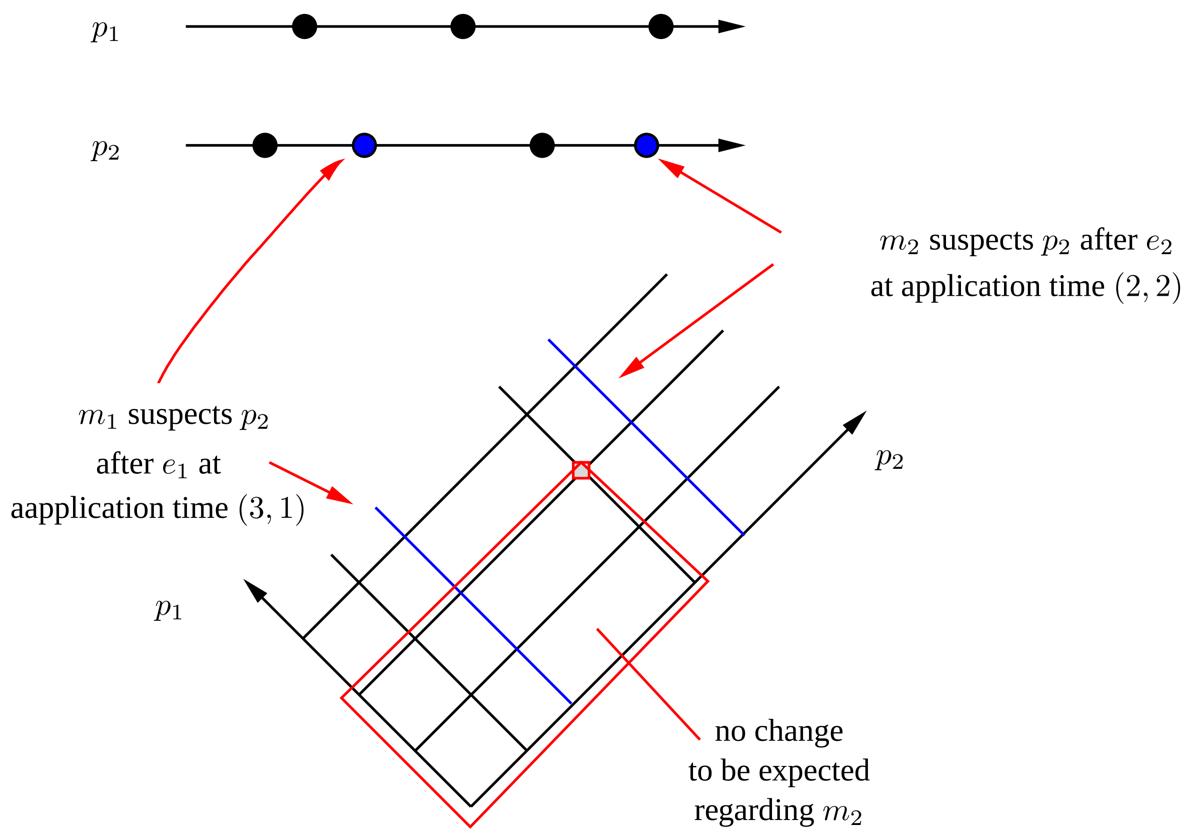
Settled region example



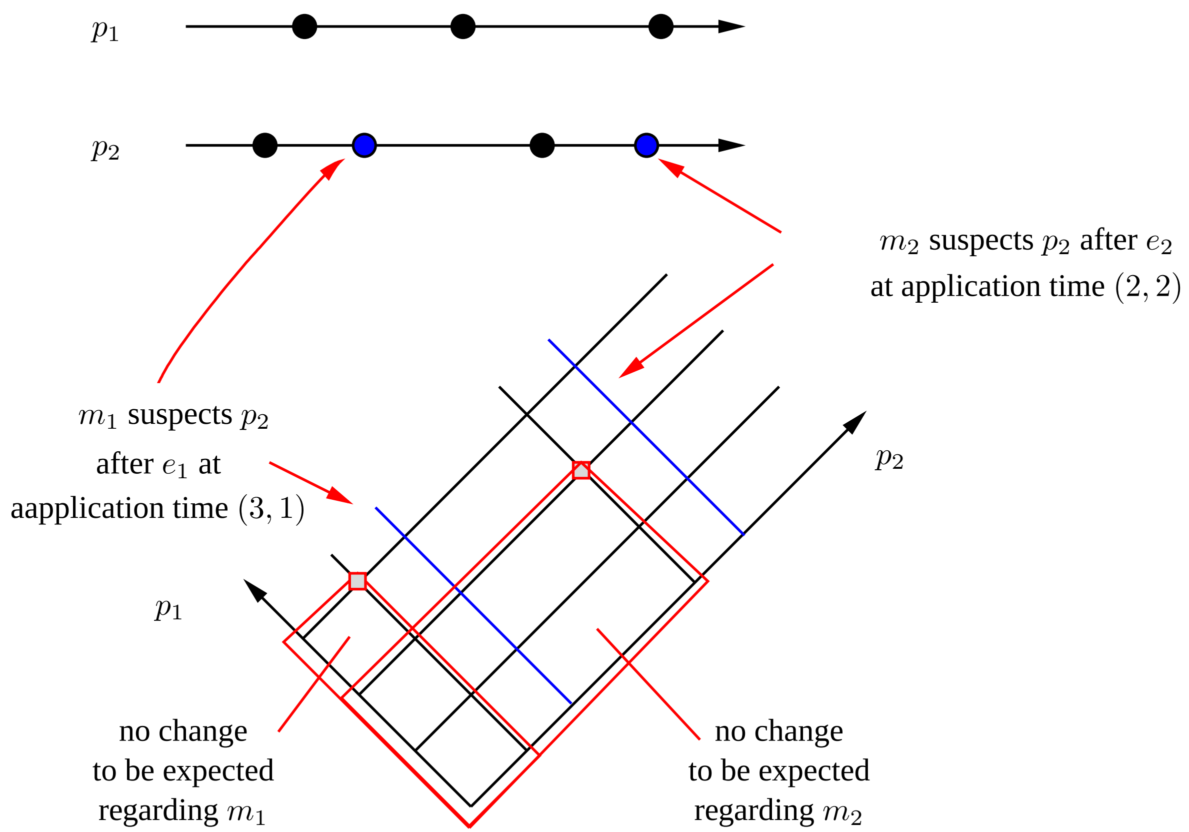
Settled region example



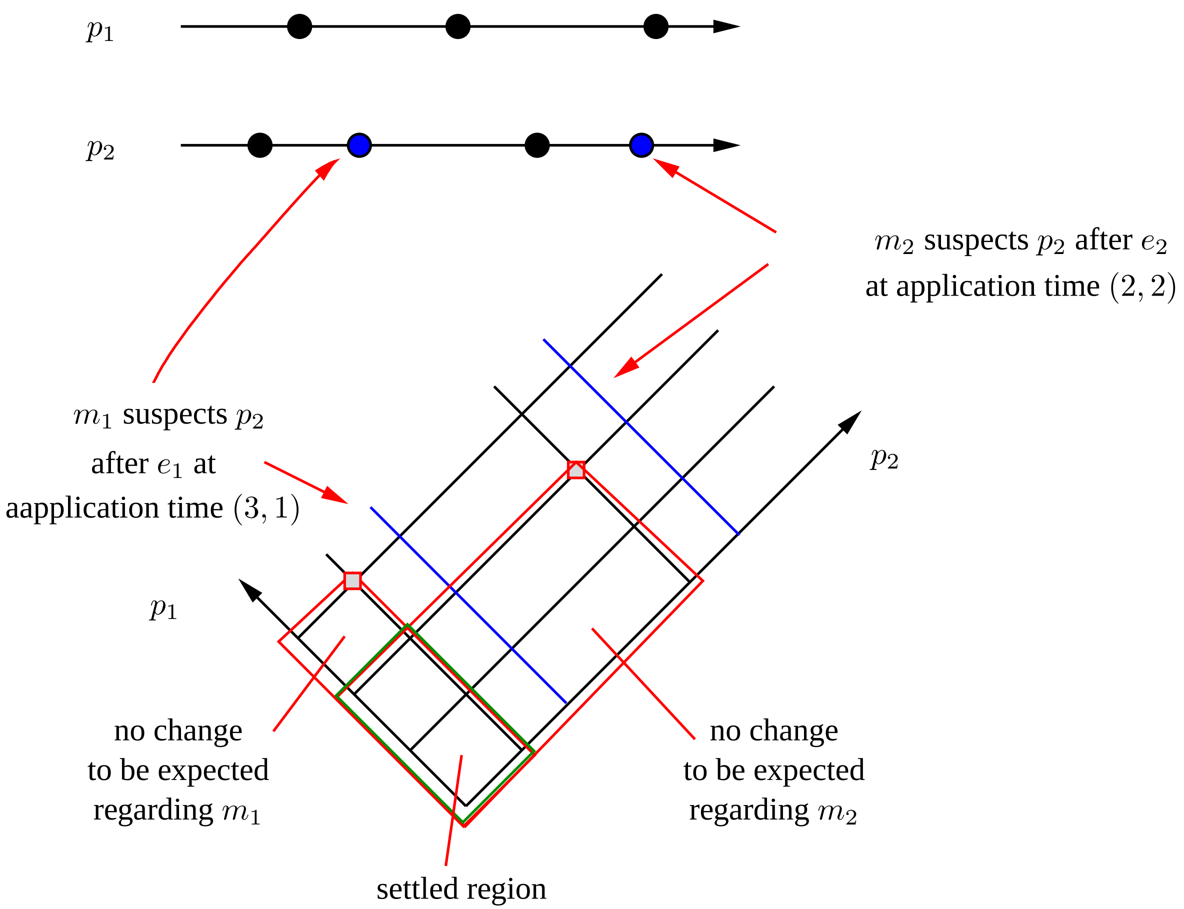
Settled region example



Settled region example



Settled region example



Advanced topics

- Algorithm works under assumption that no monitors fail.
- If monitors can fail, detection becomes harder:
  - Can still detect *negotiably* without a stable region.
  - Detection *discernibly* impossible, because accurate failure detection is needed.
  - A weaker variant (*t-discernably*) can be detected at the price of having a majority of correct monitors.

Complexity and restricted predicates

- Complexity:
  - general predicate detection is NP-complete [1].
  - Our detection algorithms are only wrappers around possibility/definitely detection.
  - Study restricted classes of predicates.
- Perfect failure detectors available:
  - No false suspicions.
  - Optimistic/pessimistic lattice are the same.
- Perfect failure detectors and crash predicates:
  - Predicates are stable.
  - $possibly{=}definitely \rightarrow negotiably{=}discernibly$

Overview of results

- First work to deal with general predicates in faulty systems.
- Observation modalities $negotiably$ and $discernibly$. . .
  - do not solve all problems in crash-affected systems.
  - reflects by their definition the inherent problem of crash failure detection.
  - can be understood in analogy to $possibly$ and $definitely$.
  - can be detected in asynchronous systems, even if monitors may crash.
- Still a lot of work to do.

References

[1] Craig M. Chase and Vijay K. Garg. Detection of global predicates: Techniques and their limitations. *Distributed Computing*, 11(4):191–201, 1998.

[2] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[3] Vijay K. Garg and J. Roger Mitchell. Distributed predicate detection in a faulty environment. In *Proceedings of the 18th IEEE International Conference on Distributed Computing Systems (ICDCS98)*, 1998.

[4] Vijay K. Garg and J. Roger Mitchell. Implementable failure detectors in asynchronous systems. In *Proc. 18th Conference on Foundations of Software Technology and Theoretical Computer Science*, number 1530 in Lecture Notes in Computer Science, Chennai, India, December 1998. Springer-Verlag.

[5] Felix C. Gärtner and Sven Kloppenburg. Consistent detection of global predicates under a weak fault assumption. In *Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems (SRDS2000)*, Nürnberg, Germany, October 2000. IEEE Computer Society Press.

Acknowledgements

- Slides produced using "cutting edge" LaTeX slide processor PPower4 by Klaus Guntermann.