Jan 30
Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanisms

Vijay Chidambaram

Global State of a distributed system

- Union of local state of nodes
- No way to get instantaneous snapshot of all nodes
- Only way to get local state of a node is by sending it a message
- How to find a meaningful global state of the system?

Why is this hard?

- Messages could be dropped or delayed
- Computed Global state may be:
  - Obsolete: state changed since we have checked
  - Incomplete: state of some nodes may be missing
  - Inconsistent: imagine a token is sent from A to B. Computed Global state may show token in both A and B

Computing Global State

- What if we simply used a lot of messages to nodes to compute global state?
- Would help with ensuring global state is complete or current (not obsolete), but will not help with ensuring global state is consistent
- Consistency cannot be achieved by throwing more messages at the problem

Global Predicate Evaluation (GPE)

- We construct a predicate based on the global state
- We want to evaluate whether this global predicate is true or false
- Examples: the system is deadlocked, a majority of nodes are alive and responsive

Modeling the system

- N sequential processes p1.. pn
- Each pair of processes has a channel
- Channels are reliable but may deliver messages out of order
- Why do we model system as async?
  - Physical delays are bounded
  - Software creates unbounded delays

Distributed Computation

- Activity distributed among N processes
- Each process sees three kinds of events:
  - Events local to that process
  - Send message to process pi
  - Receive message from process pj
- All events are recorded in the local history of the process
- Global history is union of histories of all participating processes

Global History

- Global history does not order all events
- An event A is only ordered with respect to event B if A happening affects B in some way
- Events are ordered using "happens-before" relationships

Lamport Clocks

- Notation: e(i, k) = kth event in process i
- e(i, j) < e(i, k) if j < k
- if e(i, j) = send(m), and e(k, l) = receive(m)
  - then e(i, j) < e(k, l)
- if e(i, j) < e(k, l) and e(k, l) < e(m, n)
  - then e(i, j) < e(m, n)
- All other events are considered concurrent
  - consider e(i,j) and e(k,l) concurrent
  - e(i, j) < e(k,l) is false
  - e(k, l) < e(i,j) is also false

Distributed Computation

- Formally, a distributed computation is a partially ordered set (poset) defined by the pair (H, ->)
- Not all events are ordered
- Events inside each process are totally ordered
- Events across processes are partially ordered

Cuts

- A cut of a distributed computation is a subset C of its global history H and contains an initial prefix of each of the local histories
- A cut can be defined by tuple (c1, c2, ..cN)
  - Process pi's last event in the cut is ci
  - P1's last event in the cut is c1
- Frontier of the cut: the set of events e(i, ci) for i=1..n (the last events included in the cut for each process)

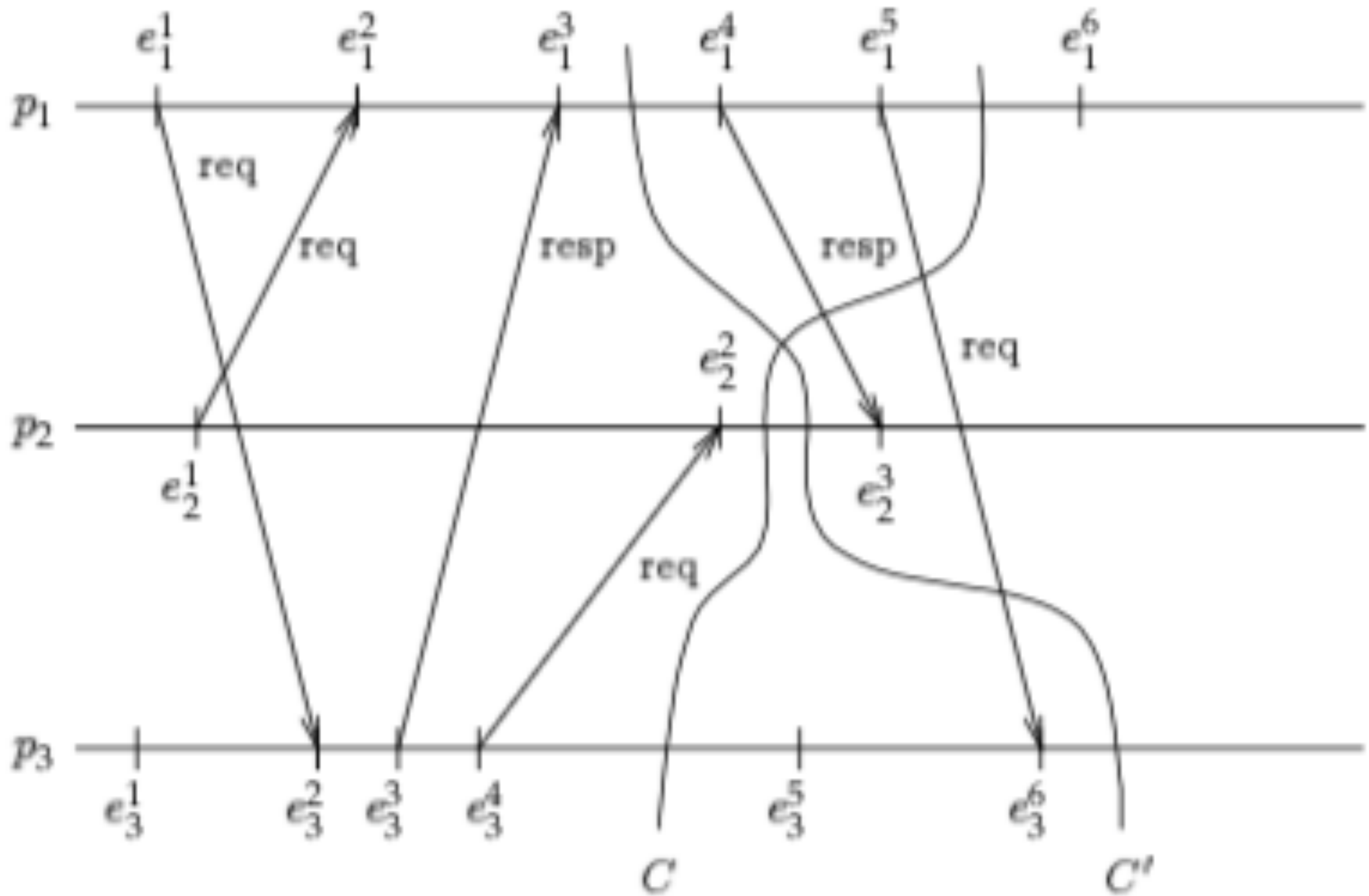

Figure 2. Cuts of a Distributed Computation

Runs

- A run of a distributed computation is total ordering R that includes all of the events in the global history and that is consistent with each local history.
- For process Pi, the events of Pi occur in the same order in R as in the history of Pi
- There are many possible runs for a single distributed computation with history H

Inconsistent Cuts

- Not all cuts are consistent
- If a cut includes receipt of a message but not the sending of the message, it is inconsistent
- More precisely, if $e(i, j) < e(k, l)$ and $e(k, l)$ is in the cut, then $e(i,j)$ must also be in the cut
- Global properties must be checked using consistent cuts (which lead to consistent global states)
- Using inconsistent cuts may lead to determination of "ghost deadlock"

Reachable States

- A run R is said to be consistent if for all events, e1 < e2 implies that e1 appears before e2 in R
- Each (consistent) global state Si of the run is obtained from the previous state Si-1 by some process executing the single event ei
- Si-1 leads to Si
- Sj is reachable from Si, if there is a series of consistent states from Si to Sj such as Si -> Sk -> Sj

Lattice

- The set of all consistent global states of a computation along with the leads-to relation defines a lattice.
- The lattice consists of n orthogonal axes, with one axis for each process
- Each path down the lattice is one run of the distributed system

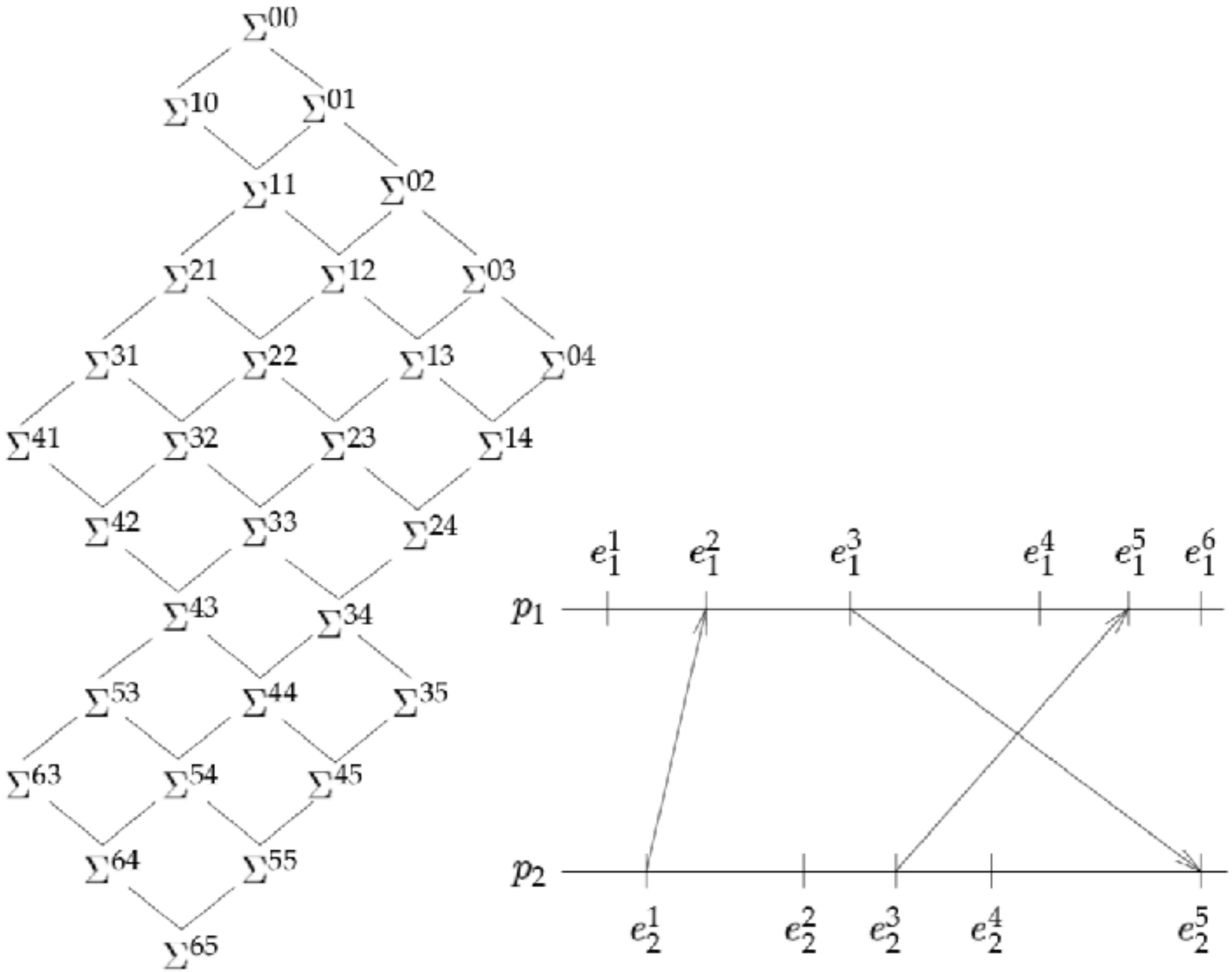

Figure 3. A Distributed Computation and the Lattice of its Global States

Observing a distributed system

- Idea 1: Active Observer
  - One monitor process sends messages to all processes
  - Constructs global state based on responses
  - Can lead to inconsistent cut

Observing a distributed system

- Idea 2: Passive Observer
  - One monitor process gets copy of all messages sent by processes
  - Different monitors observe different cuts (and hence different global states)
  - Consistent observation leads to a consistent run

Observing a distributed system

- We ensure messages from the same process are delivered in order (FIFO delivery)
  - Implemented using per-process sequence numbers
- Delivery Rule:
  - if $e1 < e2$, then $ts(e1) < ts(e2)$.
  - Deliver messages in timestamp order