Ch 9 Goals

- Introduce Byzantine Faults
- Define pulse synchronization
- Show equivalence between solving clock synchronization and pulse synchronization
- Present a fault tolerant pulse synchronization algorithm
- Show basic lower bounds on the fraction of Byzantine faults that can be tolarated.

Byzantine Faults

A **Byzantine** faulty node is a node that may behave arbitrarily.

That is, such a node does not need to follow any algorithm prescribed by the system designer.

An algorithm is **resilient** to f Byzantine faults if its performance guarantees hold for any execution in which there are at most f Byzantine faulty nodes.

In the following, for a network G=(V,E) and a set F of faulty nodes, we denote by V_g the set of correct nodes.

Pulse synchronization goals:

For each $i \in \mathcal{N}$, $v \in V_g$ generate pulse i exactly once,
($p_{v,i}$ is the time when v generates pulse i),
such that there exists S, P_{min}, P_{max}, satisfying:

1) $\sup_{i \in \mathcal{N},\, v,w \in Vg}\{|p_{v,i}-p_{w,i}|\} = S$ (skew)
2) $\inf_{i \in \mathcal{N}}\{\min_{v,\in Vg}\{p_{v,i+1}\}-\max_{v,\in Vg}\{ p_{v,i}\} \} \geq P_{min}$
3) $\sup_{i \in \mathcal{N}}\{\max_{v,\in Vg}\{p_{v,i+1}\}-\min_{v,\in Vg}\{ p_{v,i}\} \} \leq P_{max}$

Thus, **pulses** are **well aligned** and **well separated** in any <u>**feasible execution**</u> (obeying drifts and message tranmission bounds)

Impossibility Claim

Theorem:

Pulse synchronization is impossible if $3 \le n \le 3f$

- we will present confusing behaviors to correct nodes

- we will prove that Byzantine behavior presents a delimma to the protocol
  - If correct nodes refuse to increase the rate at which pulses are generated, skew will be violated.
  - If they increase the rate at which pulses are generated, P_{min} will be violated.

Breakout Room

Exchange ideas how Byzantine faults can fool us

why the case of n=2 is left out; i.e., is there an alg for n=2, f =1?

Observations

Theorem:

Pulse synchronization is impossible if $3 \le n \le 3f$

- Assume to the contrary that there is such an algorithm $\mathcal{A}$.
- We have no clue how $\mathcal{A}$ operates.
- $\mathcal{A}$ needs to gurantee the properties in any feasible execution.
- We know that for the same sequence of messages arriving at the same local hardware clock times the algorithm produces the same stream of messages and pulses.

The Art of Impossibility Results

- It is always a **dance** between finding algorithm and failing to find one
- Focus on the main difficulty in finding the protocol – which conflicting tradeoffs need to be addressed
- Simplify the model as much as you can – since you need to point out one case at which it is impossible
- Build fooling scenarios – keeping parts of the system that can't see the difference, for any possible algorithm.

- CAREFULLY check that you do not fool yourself ☺

Simulating an Algorithm

- For a given deterministic algorithm $\mathcal{A}$:
- Assuming we have control of all nodes'
  - initial state and initialization times,
  - all local hardware clock drifts,
  - and the transmission time of each message.

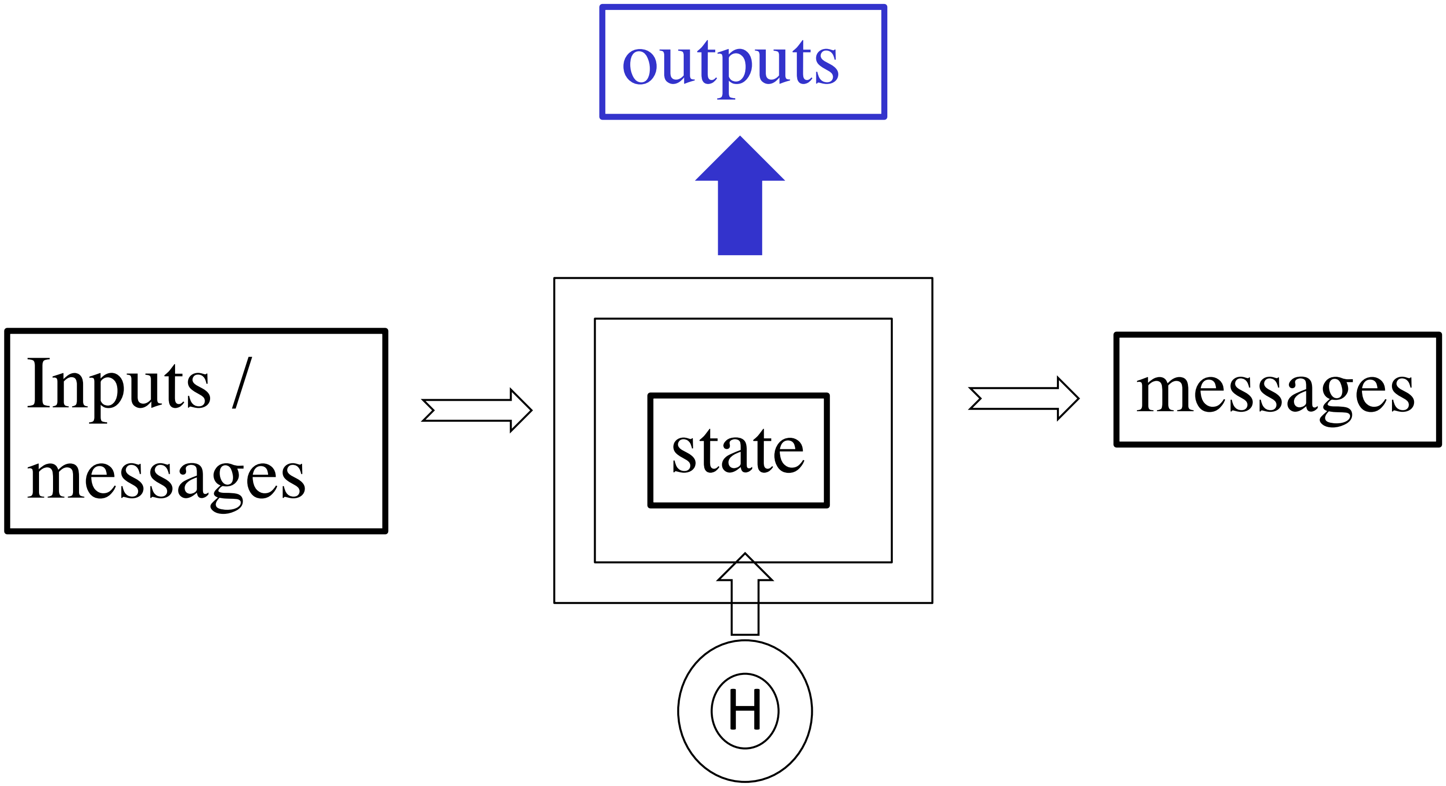
Will we know all the messages that will be exchanged?

- For example: Let all $H_v(0)$ be 0. Do we know what is the first message each node sends?

- If we know at which local clock time these messages will be received, will we know which messages will be sent next?

.

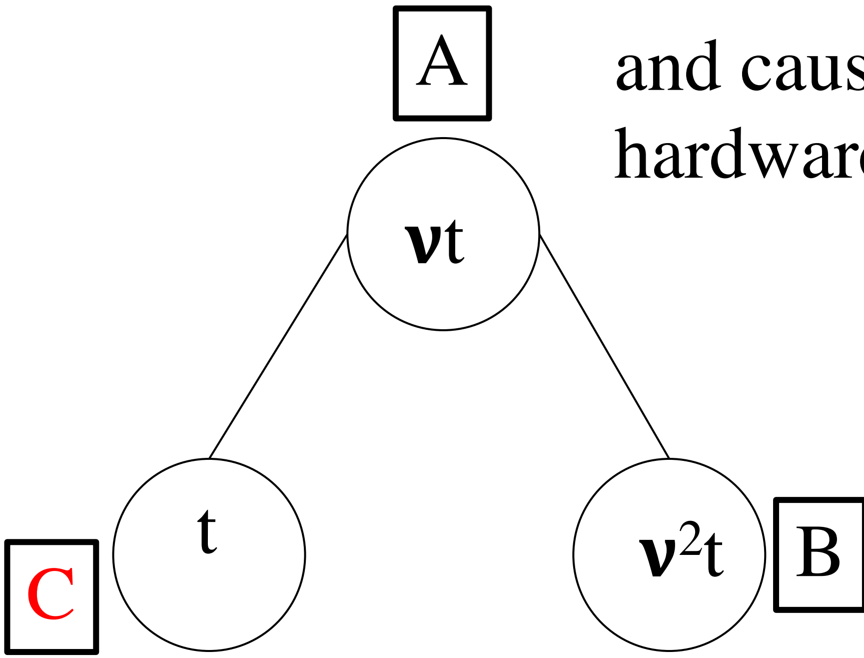
A State Machine



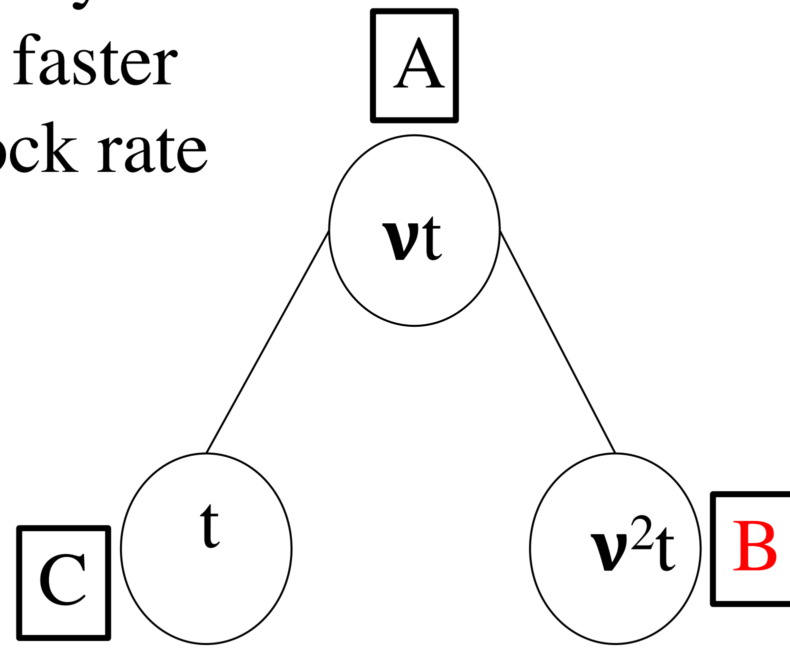
The sequence of messages and outputs depends **solely** on

1. the initial state and initial input (**fixed -known**)
2. the sequence of messages and inputs it receives
3. hardware clock readings

Disconnecting the Link Between B and C



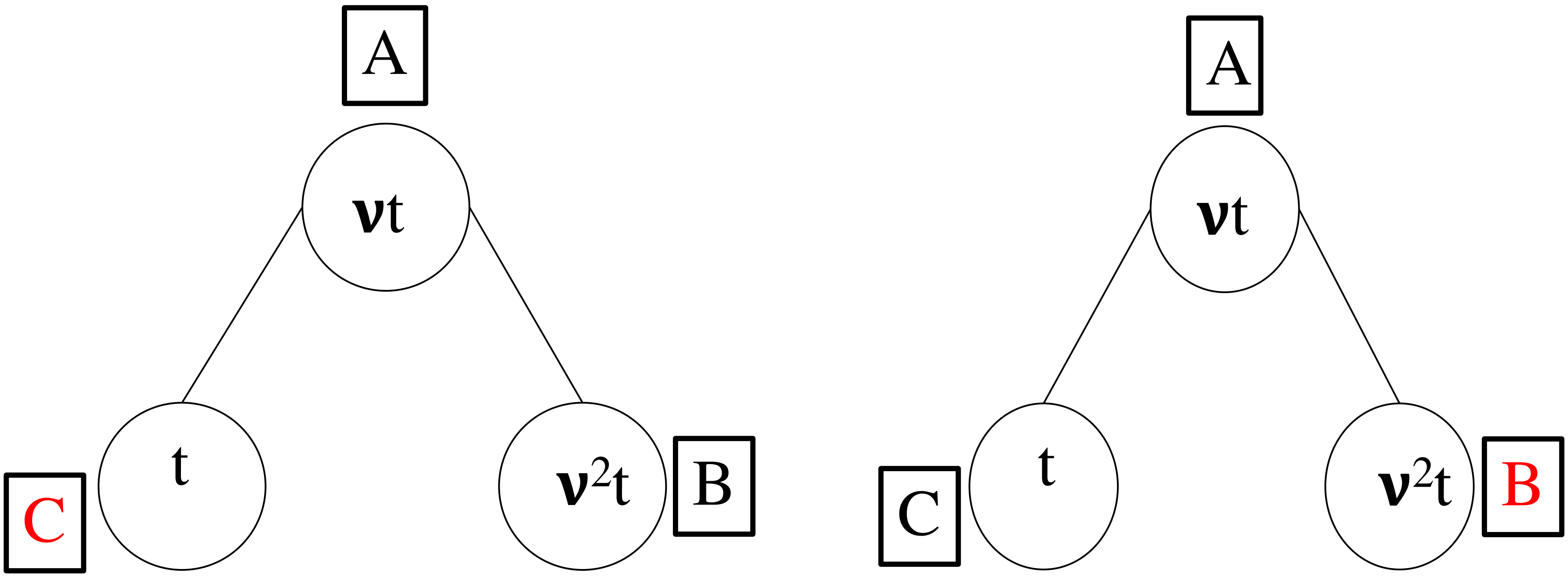
$\boldsymbol{\nu}>1$ is relatively small and causes a faster hardware clock rate



Two scenarios: In both the faulty one behaves as though the link is disconnected.

Algorithm $\boldsymbol{\mathcal{A}}$ instructs correct nodes what to send and when to produce **pulses**.

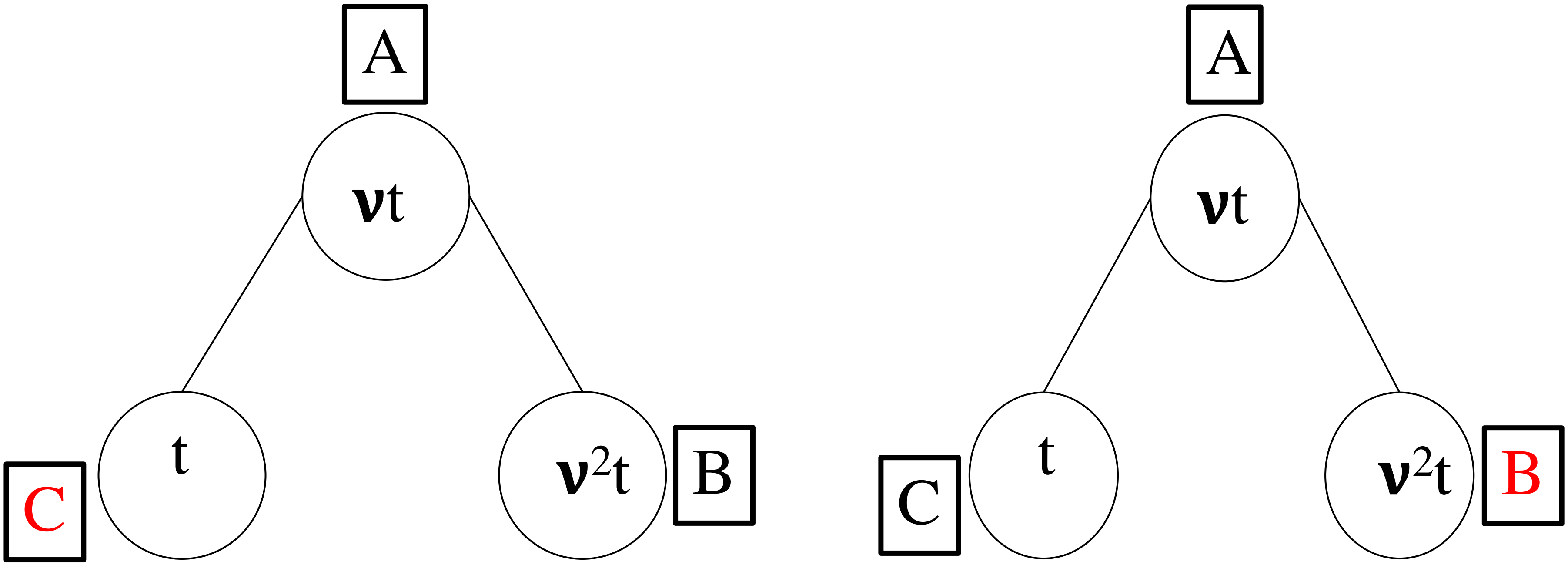
Disconnecting the Link Between B and C



Assume that in both scenarios all messages arrive at the identical clock times – the messages exchanged are identical in both executions.

Node A cannot tell the difference between the two.

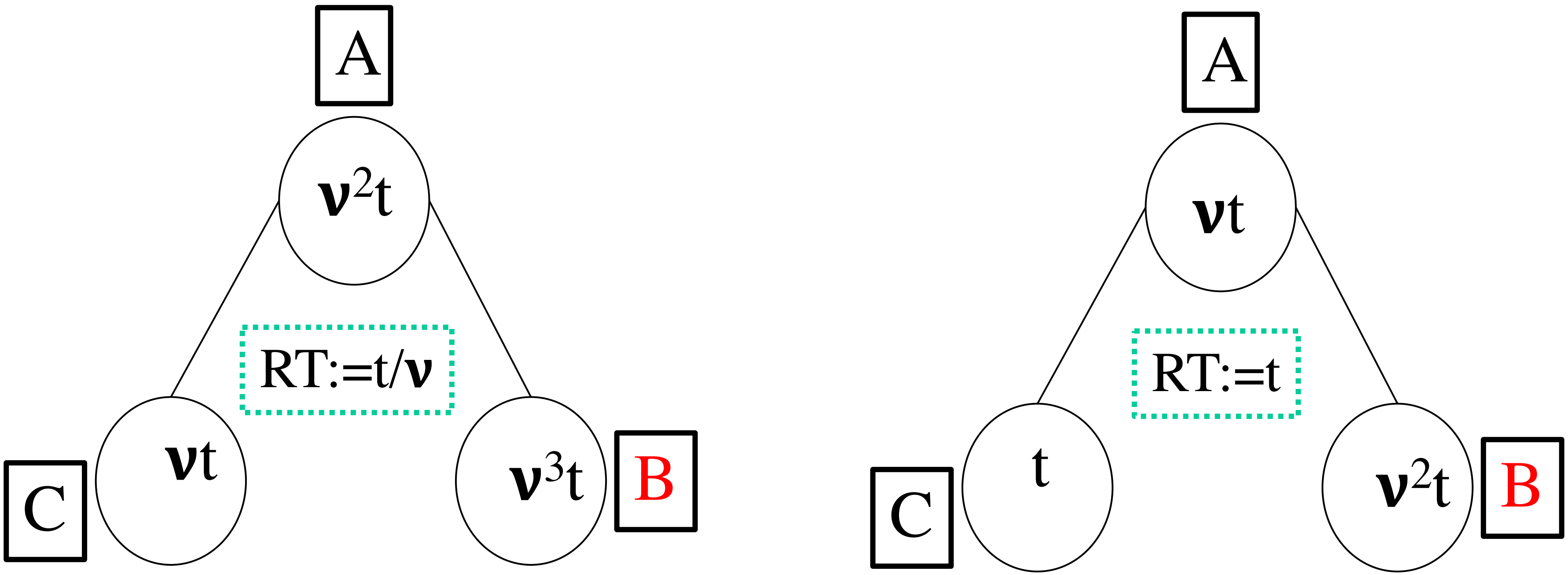
Disconnecting the Link Between B and C



Given that we know the clock drifts and the clock time of receiving messages

we can simulate the whole message exchange.

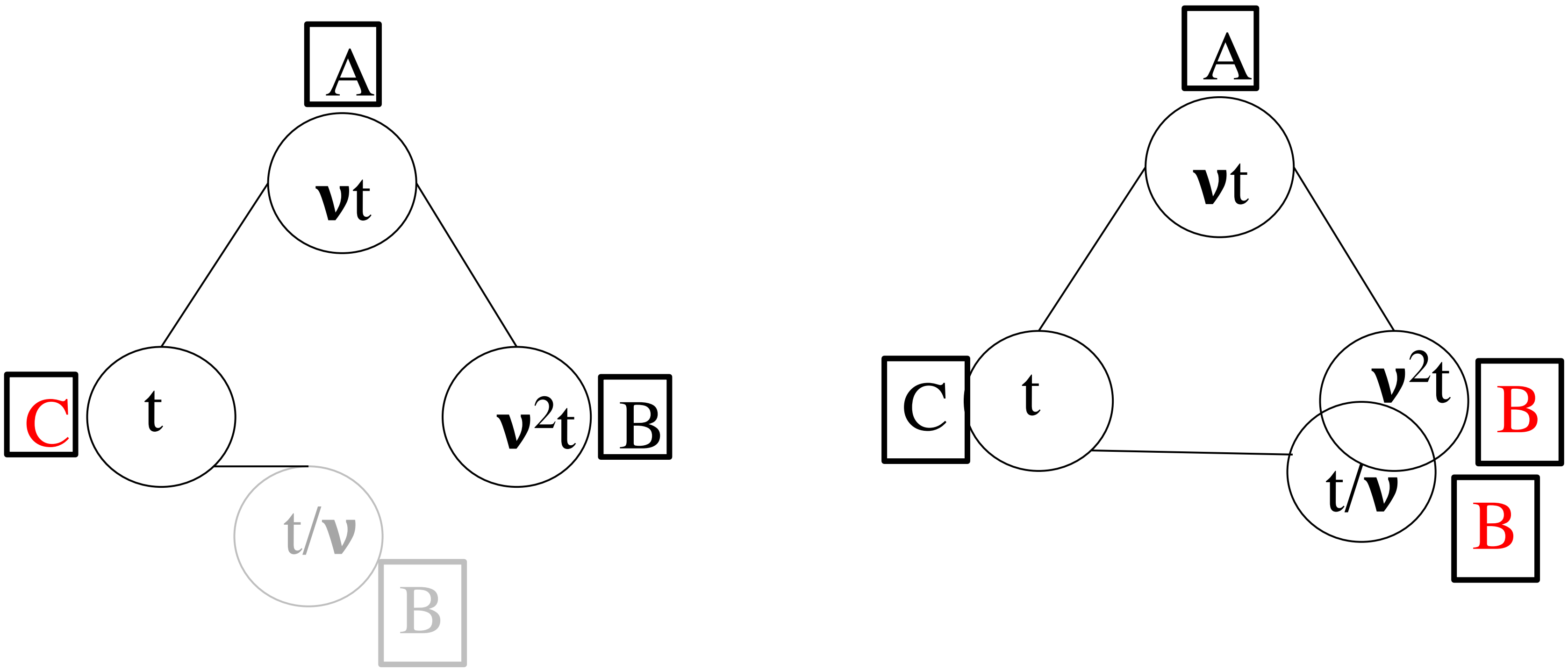
Alternative Real Times



Nodes do not have access to external real-time

No correct node can tell the difference

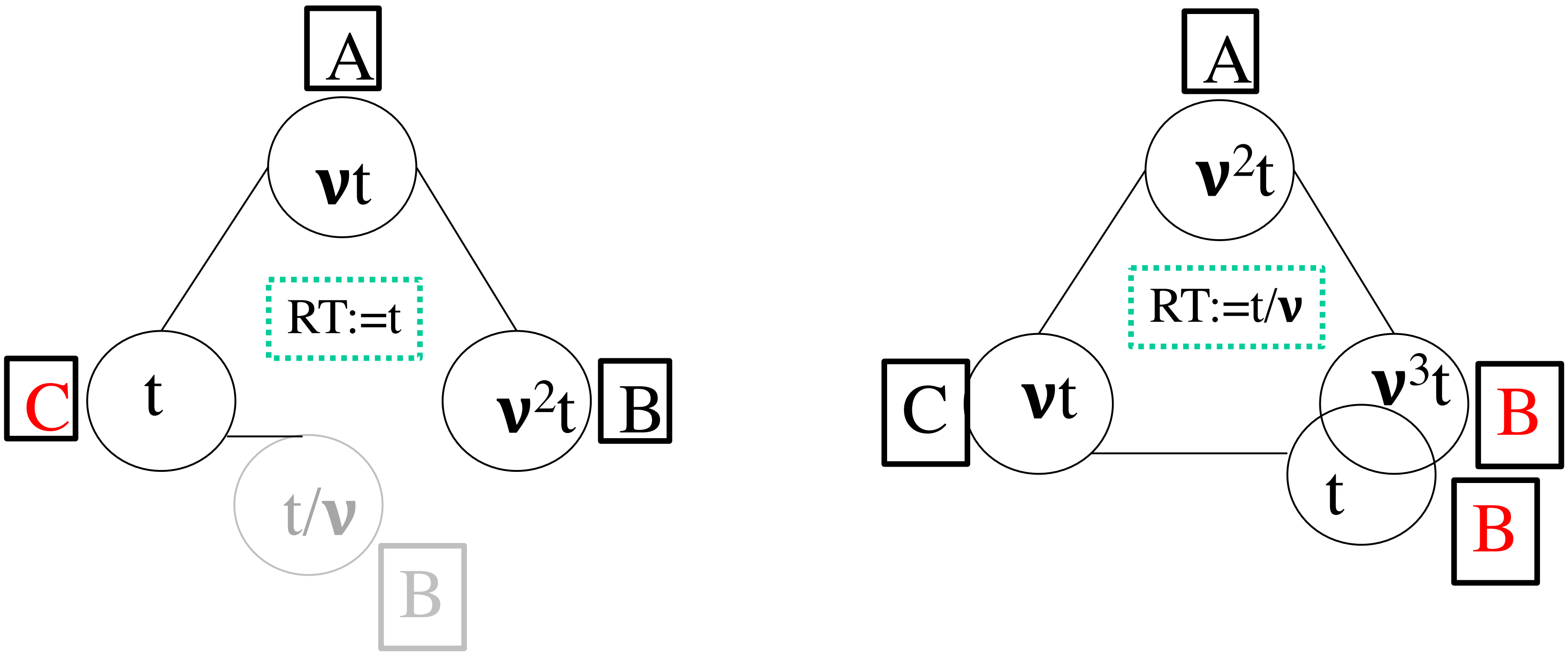
Fake News



Assume that in both scenarios all messages arrive at the identical clock times –

Node A can't tell the difference. The protocol instructs the correct nodes what to send and when to "pulse"

Fake News - Alternatives



Assume that in both scenarios all messages arrive at the identical clock times – B can envision receiving …

Node A can't tell the difference. The protocol instructs the correct nodes what to send and when to "pulse"

The Rules of the Game

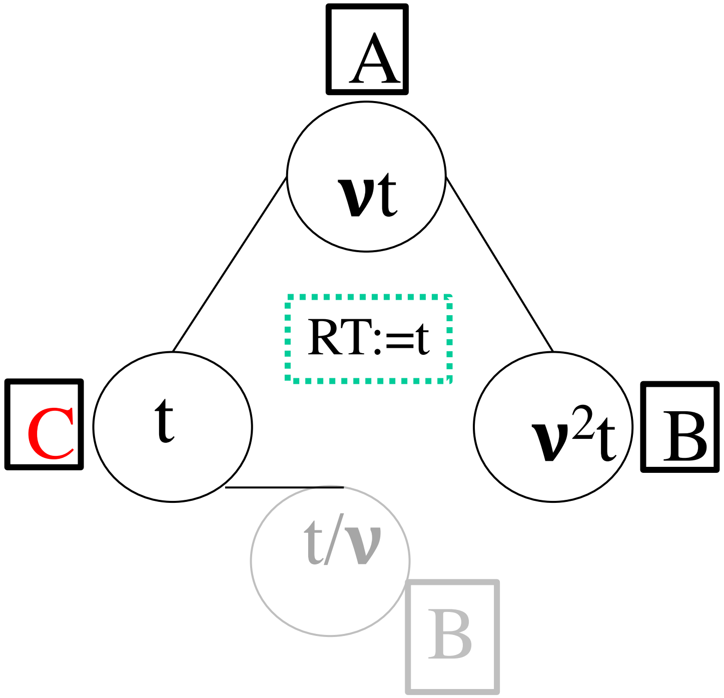
- For a given deterministic algorithm $\mathcal{A}$:
- $\mathcal{A}$ needs to provide the pulse synchronization guarantees in EVERY feasible execution.

- It is enough that it fails on a single feasible execution to prove the impossibility result.

- We can choose specific clock drifts and message transmissions in a <u>feasible execution</u> and also choose which nodes to fail and instruct them how to behave.

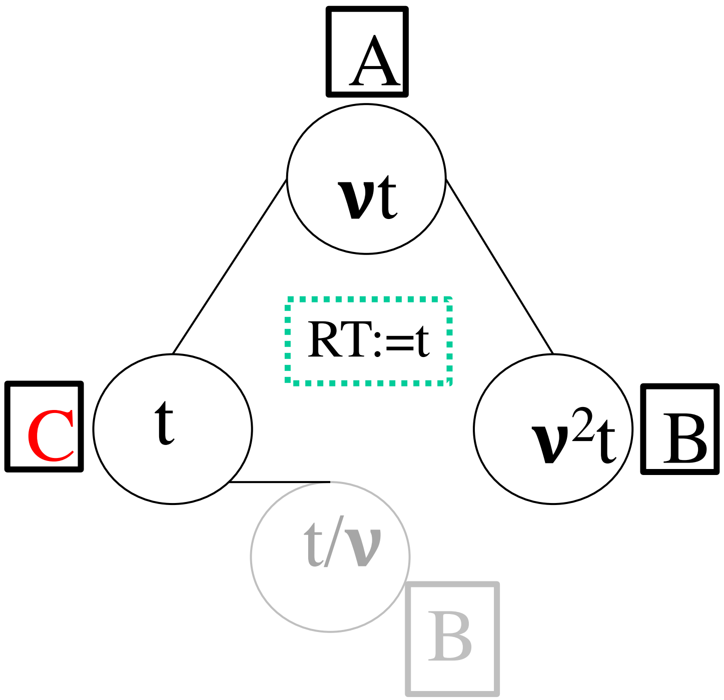
- We will construct a general scheme that identifies a feasible execution at which such an algorithm $\mathcal{A}$ fails, provided that the algorithm guarantees a bounded skew.

Execution $\mathcal{E}_0$

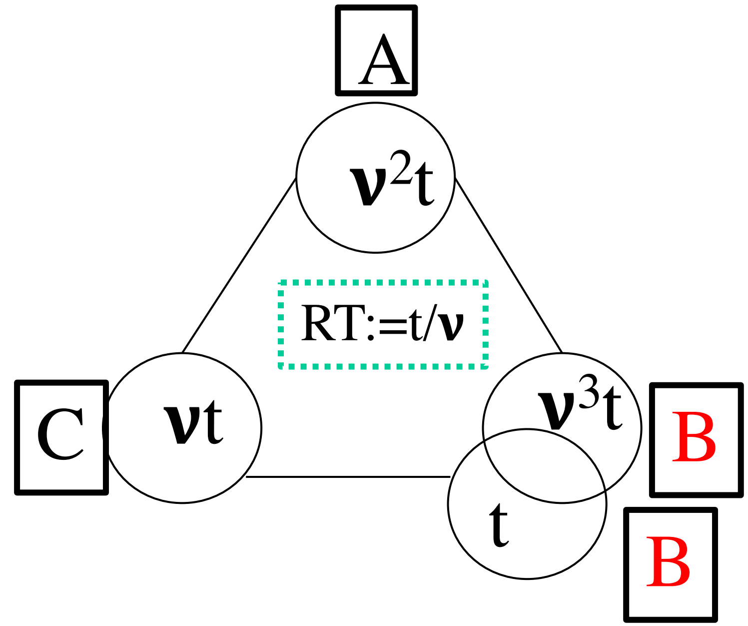


The initial execution $\mathcal{E}_0$.
The real time rate is 1. Set $\boldsymbol{\nu}^3 = \vartheta$. (we ignore **u**)
All messages arrive within **d** real time: a message sent to w at t, is received by w at time $H_w(t)+d$.
Algorithm $\mathcal{A}$ does not have access to real-time.

Execution $\mathcal{E}_0$



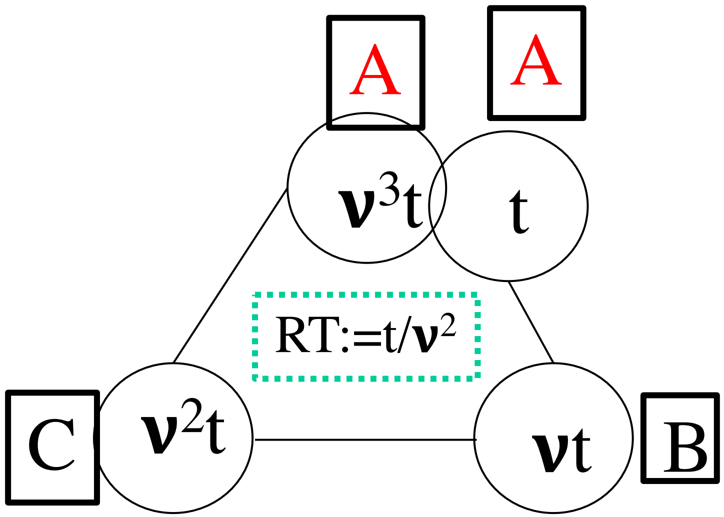
Execution $\mathcal{E}_1$



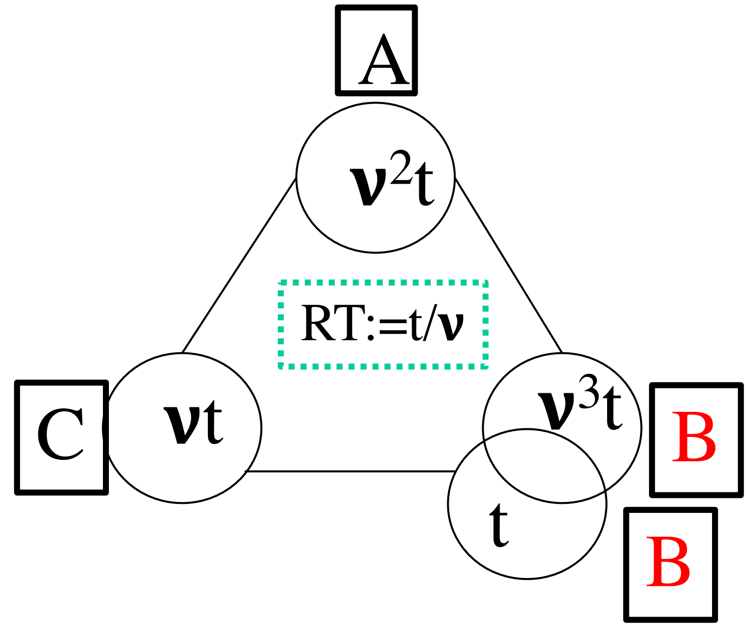
In both executions all messages arrive at identical clock times –

Node A can't tell the difference. The protocol instructs the correct nodes what to send and when to "pulse"

Execution $\mathcal{E}_2$



Execution $\mathcal{E}_1$



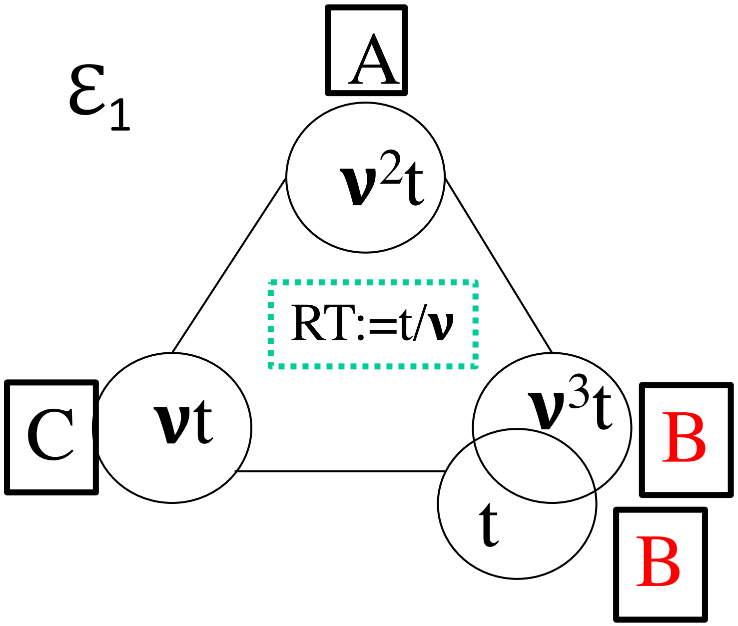
In both executions all messages arrive at identical clock times –

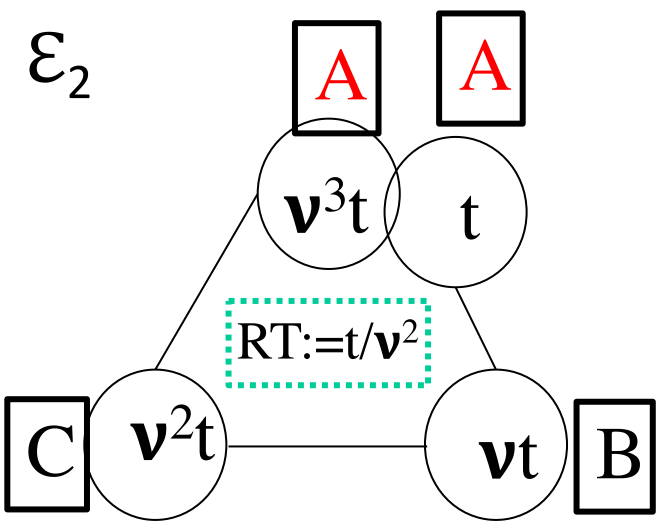
Node C can't tell the difference. The protocol instructs the correct nodes what to send and when to "pulse"

Lemma 9.12

Suppose $3 \le n \le 3f$. For any pulse synchronization algorithm $\mathcal{A}$, there exists $\boldsymbol{\nu} > 1$, such that in every two consecutive executions, there is a correct node that can't distinguish between them.

- We choose (ignoring u) $\boldsymbol{\nu}^3 = \vartheta$. Hardware clocks are by a factor of $\boldsymbol{\nu}$ apart, so are within the feasible bounds.
- The local times at which message are received is determined to be: if v send a message to w at time t, it is received by w at time $H_w(t)+d$.
- The real time in execution $\mathcal{E}_i$ is by a factor of $\boldsymbol{\nu}$ faster that the real time time in execution $\mathcal{E}_{i+1}$.

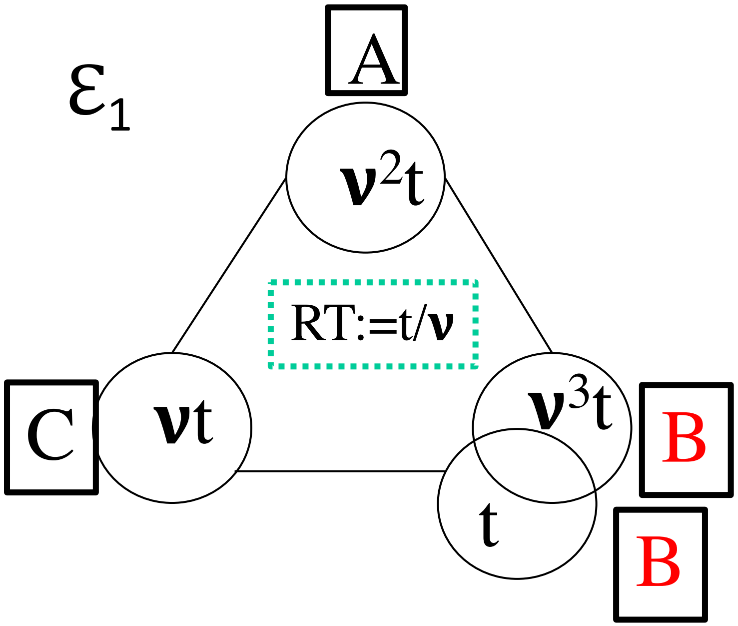




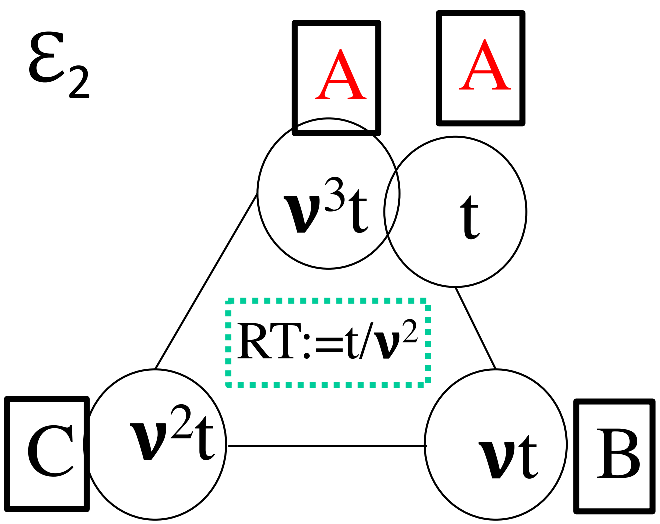
	$H_A(t)$	$H_B(t)$	$H_C(t)$
$\mathcal{E}_0$	νt	$\nu^2 t$	$\leftarrow$ arbitrary $t \rightarrow$
$\mathcal{E}_1$	$\nu^2 t$	$\leftarrow \nu^3 t$ $t \rightarrow$	νt
$\mathcal{E}_2$	$\leftarrow \nu^3 t$ $t \rightarrow$	νt	$\nu^2 t$
$\mathcal{E}_3$	νt	$\nu^2 t$	$\leftarrow \nu^3 t$ $t \rightarrow$
$\mathcal{E}_4$	$\nu^2 t$	$\leftarrow \nu^3 t$ $t \rightarrow$	νt
$\mathcal{E}_5$	$\leftarrow \nu^3 t$ $t \rightarrow$	νt	$\nu^2 t$
$\mathcal{E}_6$	νt	$\nu^2 t$	$\leftarrow \nu^3 t$ $t \rightarrow$
$\dots$	$\dots$	$\dots$	$\dots$

We can see that C can't tell the difference

$\mathcal{E}_1$



$\mathcal{E}_2$



	$H_A(t)$	$H_B(t)$	$H_C(t)$
$\mathcal{E}_0$	νt	$\nu^2 t$	$\leftarrow$ ~~arbitrary~~ $t \rightarrow$
$\mathcal{E}_1$	$\nu^2 t$	$\leftarrow \nu^3 t$ $t \rightarrow$	νt
$\mathcal{E}_2$	$\leftarrow \nu^3 t$ $t \rightarrow$	νt	$\nu^2 t$
$\mathcal{E}_3$	νt	$\nu^2 t$	$\leftarrow \nu^3 t$ $t \rightarrow$
$\mathcal{E}_4$	$\nu^2 t$	$\leftarrow \nu^3 t$ $t \rightarrow$	νt
$\mathcal{E}_5$	$\leftarrow \nu^3 t$ $t \rightarrow$	νt	$\nu^2 t$
$\mathcal{E}_6$	νt	$\nu^2 t$	$\leftarrow \nu^3 t$ $t \rightarrow$
...	...	...	...

Messages to be sent by a faulty node are well defined

Theorem 9.13 – pulse synchronization is impossible for $3 \le n \le 3f$

We will show that if algorithm $\mathcal{A}$ ensures a bounded skew it needs to produce pulses faster and faster as we move from one execution to the other.

- there exists $v \in V_g^{\mathcal{E}_i}$ such that

$$p_{v,j}^{(\mathcal{E}_i)} - p_{v,1}^{(\mathcal{E}_i)} \le (j-1)\nu^{-i} P_{\max} + 2i\mathcal{S}$$

- let v be the one for $\mathcal{E}_i$ and w be the correct in both $\mathcal{E}_i$ and $\mathcal{E}_{i+1}$
- by the skew bound

$$p_{w,j}^{(\mathcal{E}_i)} - p_{w,1}^{(\mathcal{E}_i)} \le p_{v,j}^{(\mathcal{E}_i)} - p_{v,1}^{(\mathcal{E}_i)} + 2\mathcal{S} \le (j-1)\nu^{-i} P_{\max} + 2(i+1)\mathcal{S}$$

- by the construction of the executions

$$p_{w,j}^{(\mathcal{E}_{i+1})} = \nu^{-1} p_{w,j}^{(\mathcal{E}_i)}$$

.

Theorem 9.13 – pulse synchronization is impossible for $3 \leq n \leq 3f$

- thus, we proved the inductive step:

$$p_{v,j}^{(\mathcal{E}_i)} - p_{v,1}^{(\mathcal{E}_i)} \leq (j-1)\nu^{-i} P_{\max} + 2i\mathcal{S}$$

- now let's choose i large enough such that

$$\boldsymbol{\nu}^{-i} P_{max} < P_{min}$$

- let v be the one for $\mathcal{E}_i$ in the induction step. Choose j large enough such that

$$j-1 > 2iS(P_{min} - \boldsymbol{\nu}^{-i} P_{max})$$

- it follows that

$$p_{v,j}^{(\mathcal{E}_i)} - p_{v,1}^{(\mathcal{E}_i)} \leq (j-1)\nu^{-i} P_{\max} + 2i\mathcal{S} < (j-1) P_{\min} .$$

- violating the P_{min} requirement.

.

Theorem 9.13 – pulse synchronization is impossible for $3 \leq n \leq 3f$

- we claimed everything for the case n=3 and f=1. So we proved it only for this case.

- To complete the proof, assume we have an algorithm for the case n>3. We will reduce the algorithm to an algorithm for the case n=3.
- We divide the n nodes into 3 groups of size up to f each.
- Each of the 3 nodes in the case of n=3 simulates one group.

- This reduction completes the proof.

The Art of Impossibility Results

- It is always a **dance** between finding algorithm and failing to find one
- Focus on the main difficulty in finding the protocol – which conflicting tradeoffs need to be addressed
- Simplify the model as much as you can – since you need to point out one case at which it is impossible
- Build fooling scenarios – keeping parts of the system that can't see the difference, for any possible algorithm.

- CAREFULLY check that you do not fool yourself ☺