A Principled Way of Designing Efficient Protocols

Yoram Moses

Technion

Partly joint with Armando Castañeda and Yannai Gonczarowski

Motivation

THEME

SIROCCO is devoted to the study of the interplay between communication and knowledge in multi-processor systems from both the qualitative and quantitative viewpoints. Special emphasis is given to innovative approaches and fundamental understanding...

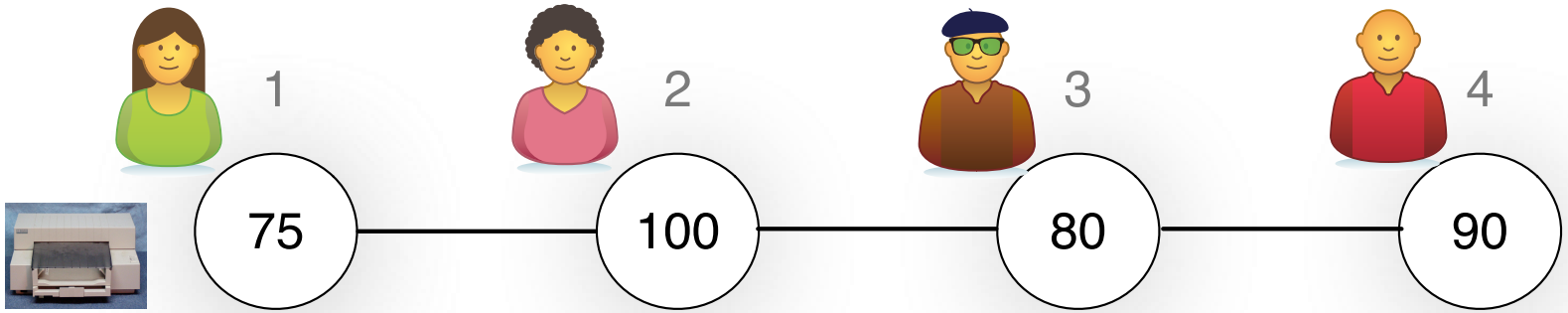
Many Models of Distributed Computing

- Communication: Message passing, shared memory, visual signalling
- Topology: Fixed, Dynamic
- Timing: Clocks, timing guarantees on actions and events (synchrony asynchrony, partial synchrony)
- Computing power: From mainframes, servers, mobile devices, low-powered sensors
- Failure modes, Uniqueness of ID's, etc...

No unifying "Turing-machine" model for distributed systems

Lack of general results that apply to "all systems"

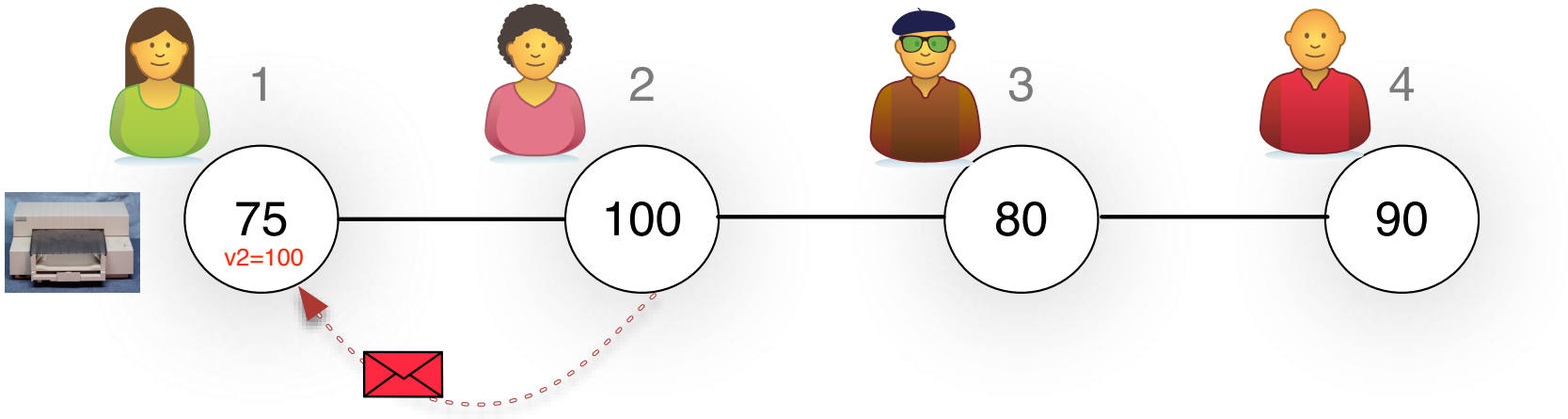
Example: Computing the Maximum (CTM)



- Each node i has an initial value v_i
- Agent 1 must print the maximal value
- After receiving "$v_2 = 100$" Agent 1 has the maximum.

  Can she act?

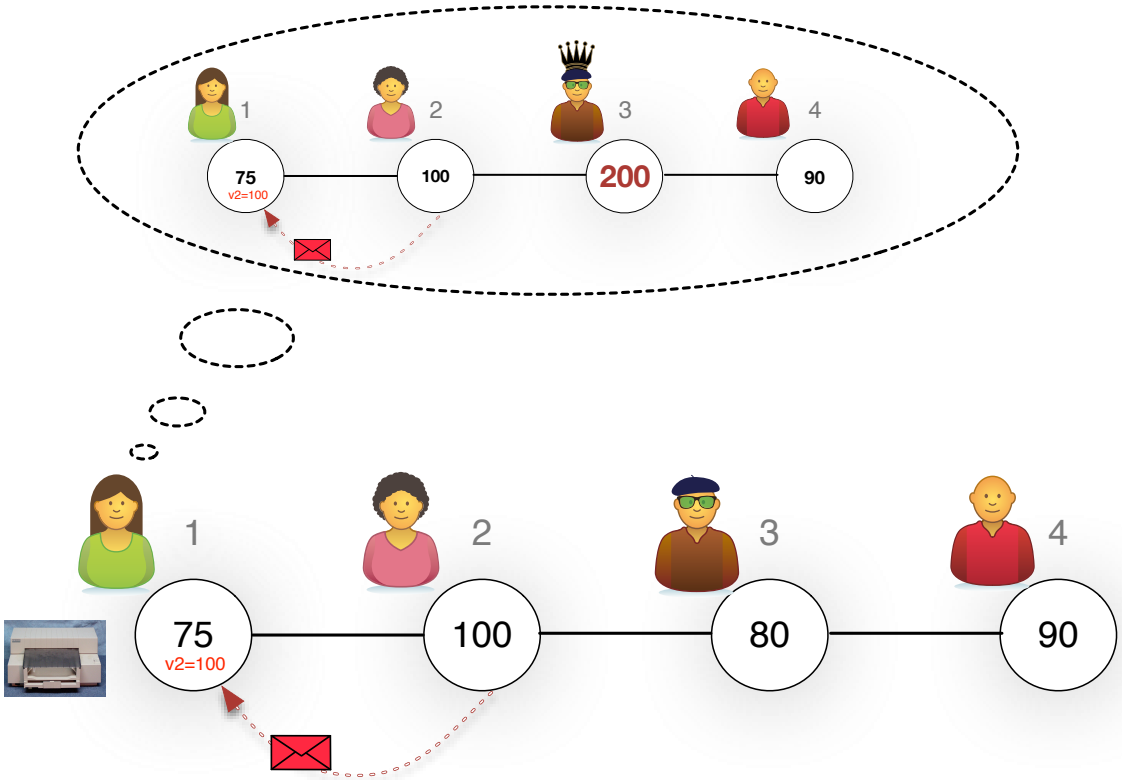
Example: Computing the Maximum (CTM)



- Each node i has an initial value v_i
- Agent 1 must print the maximal value
- After receiving "$v_2 = 100$" Agent 1 has the maximum.

  Can she act?

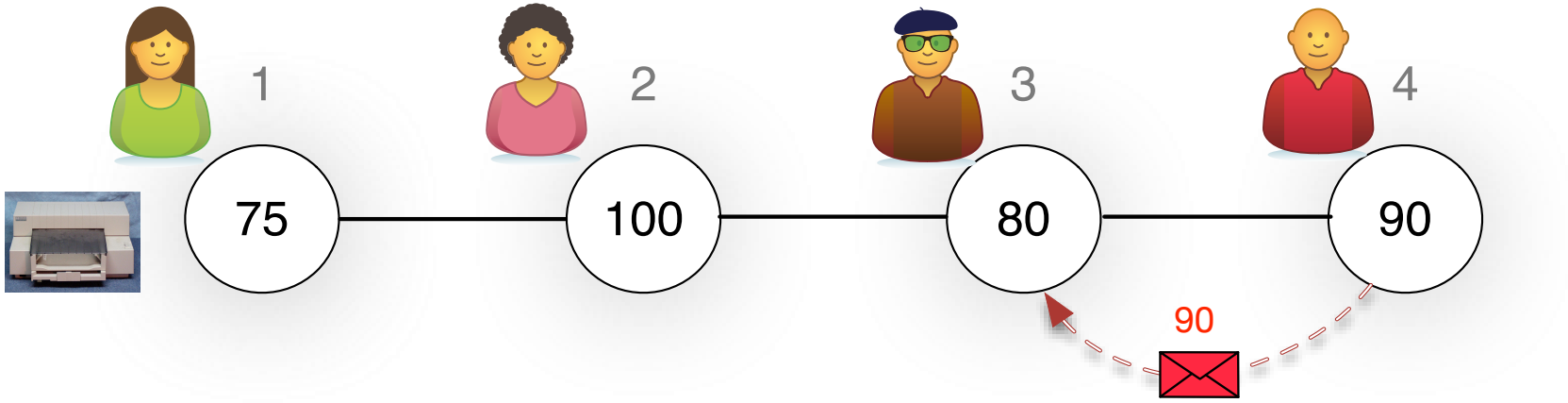
Example: Computing the Maximum (CTM)



- Each node i has an initial value v_i
- Agent 1 must print the maximal value
- After receiving "$v_2 = 100$" Agent 1 has the maximum.

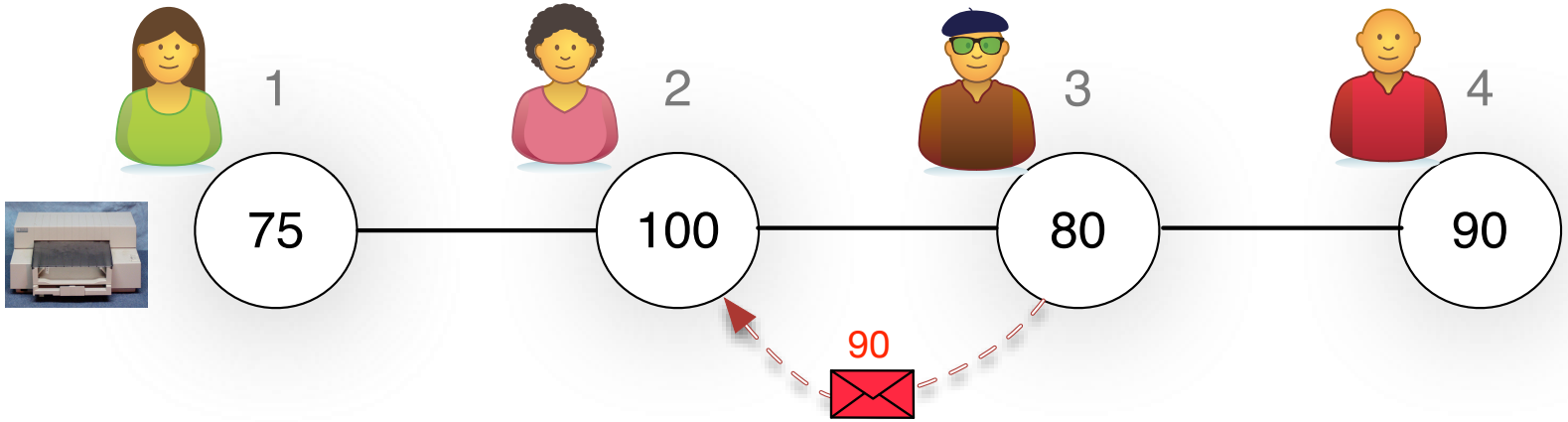
  Can she act?

Collecting Values



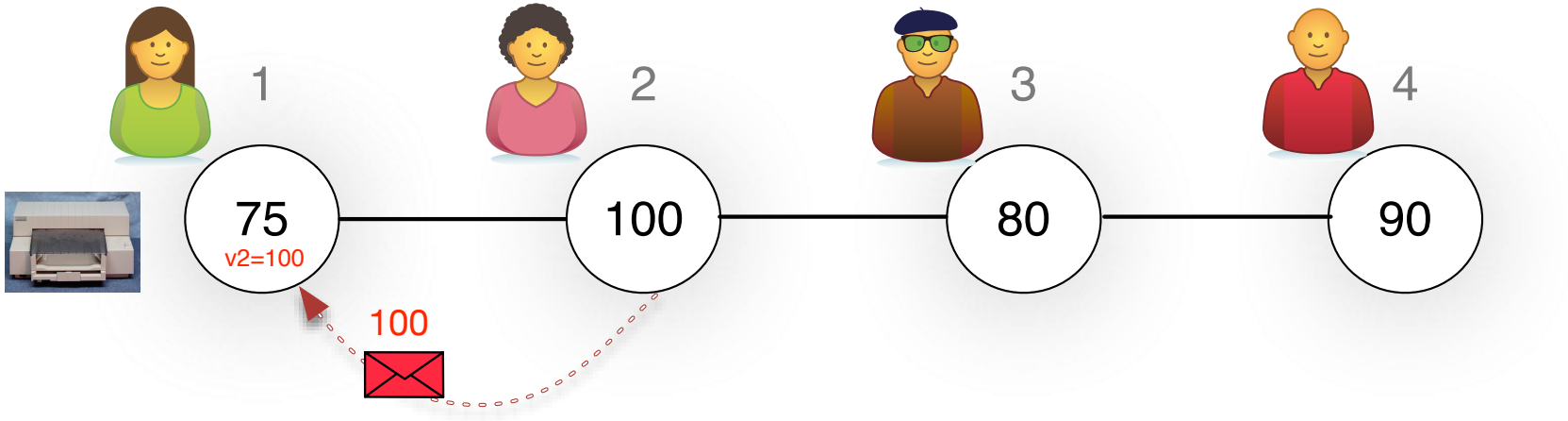
- Collecting all values is not necessary
- Collecting all values is not sufficient:
  Alice might not know that she has all values

Collecting Values



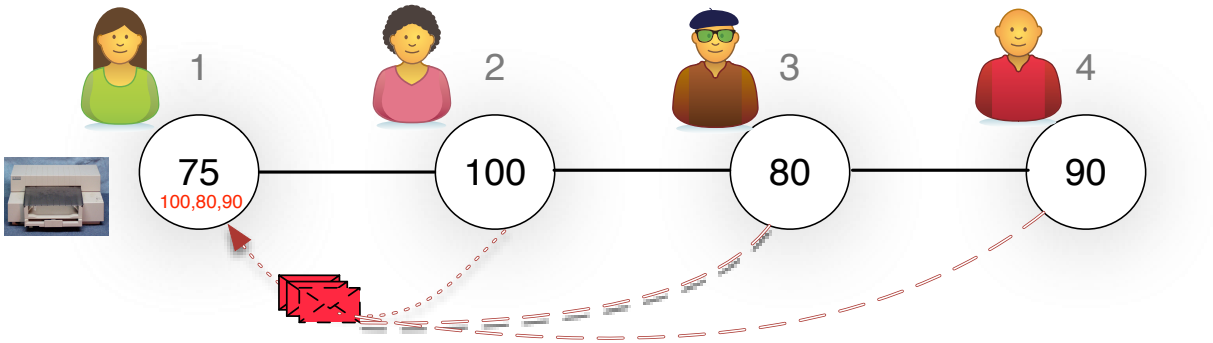
- Collecting all values is not necessary
- Collecting all values is not sufficient:
  Alice might not know that she has all values

Collecting Values



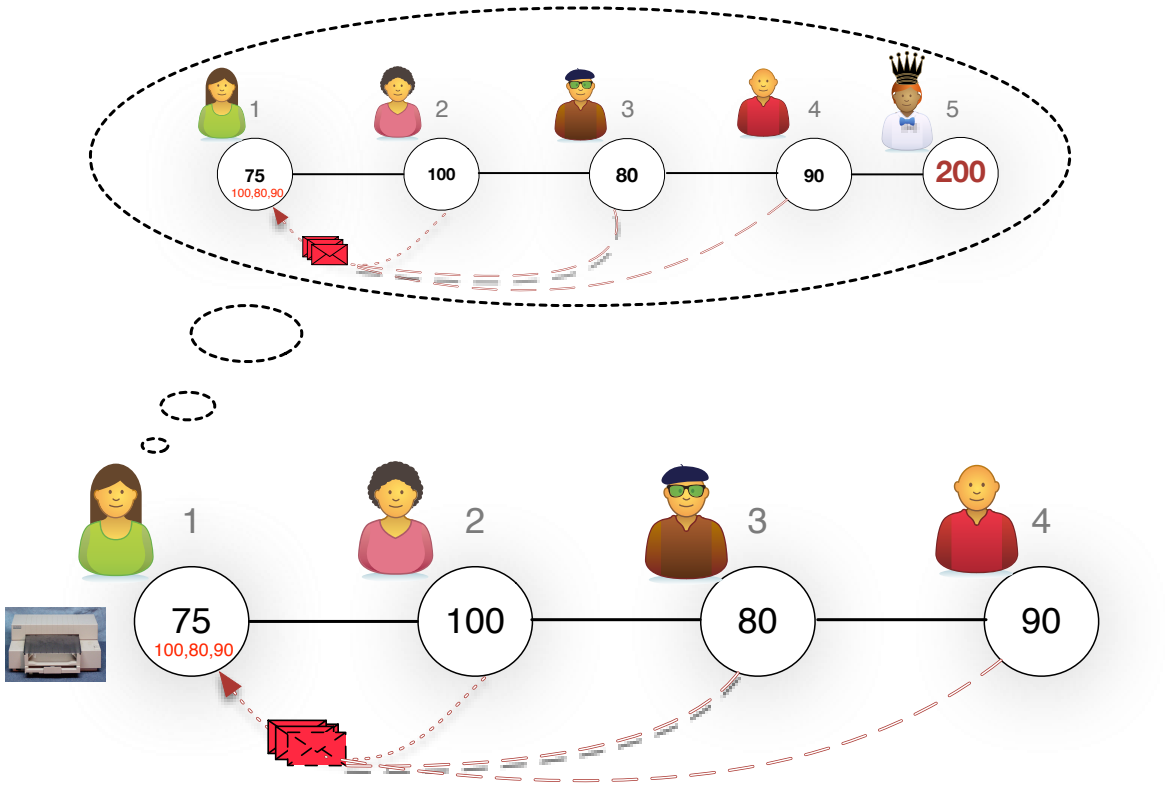
- Collecting all values is not necessary
- Collecting all values is not sufficient:
  Alice might not know that she has all values

Collecting Values



- Collecting all values is not necessary
- Collecting all values is not sufficient:
  Alice might not know that she has all values

Collecting Values



- Collecting all values is not necessary
- Collecting all values is not sufficient:
  Alice might not know that she has all values

What is CTM about if not collecting values?

Knowledge

Knowing that *Max* = *c* is necessary and sufficient for printing *c*.

Knowing that $Max = c$ can depend on:

- Messages received
- The agents' protocol
- The domain of possible initial values
- The network topology
- Timing guarantees re: communication, synchrony, activation
- Possibility of failures, ...

Needing to know the maximum is an instance of a general principle

The **Knowledge of Preconditions** Principle (**KoP**)

If φ must be true when i performs α

Then $K_i\varphi$ must be true when i performs α

Then $K_i\varphi$ is a prerequisite for i performing α

The **Knowledge of Preconditions** Principle (**KoP**)

If φ must be true when i performs α

Then $K_i\varphi$ must be true when i performs α

If good credit is a prerequisite for ATM payment

Then K_{atm}(credit) is a prerequisite for ATM payment

The **Knowledge of Preconditions** Principle (**KoP**)

If φ must be true when i performs α

Then $K_i\varphi$ must be true when i performs α

If Empty Critical Section is a prerequisite for i entering the CS

Then K_i(empty CS) is a prerequisite for i entering the CS

This is useful for analyzing Mutual Exclusion [M.&Patkin 2015]

The **Knowledge of Preconditions** Principle (**KoP**)

If φ must be true when i performs α

Then $K_i\varphi$ must be true when i performs α

If Alice has moved is a prerequisite for Bob's move

Then K_{Bob}(Alice has moved) is a prerequisite for Bob's move

The **Knowledge of Preconditions** Principle (**KoP**)

If φ must be true when i performs α

Then $K_i\varphi$ must be true when i performs α

$\Longrightarrow$ All standard specifications are epistemic:

Knowledge is a prerequisite for action

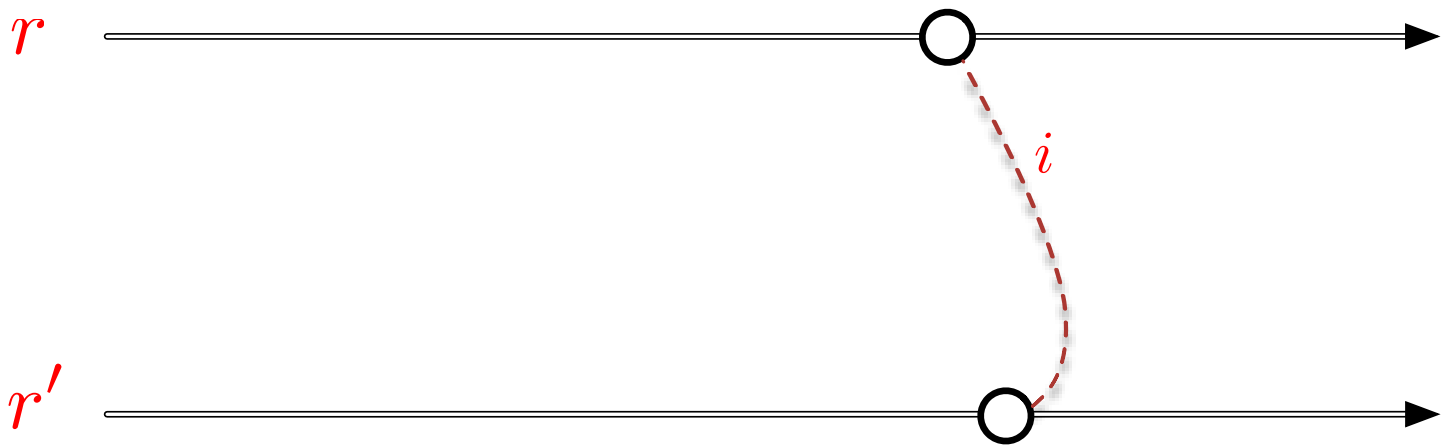
This is a fundamental theorem of multi-agent systems

A Theory of Knowledge in Distributed Systems

A three decades old theory of knowledge is based on

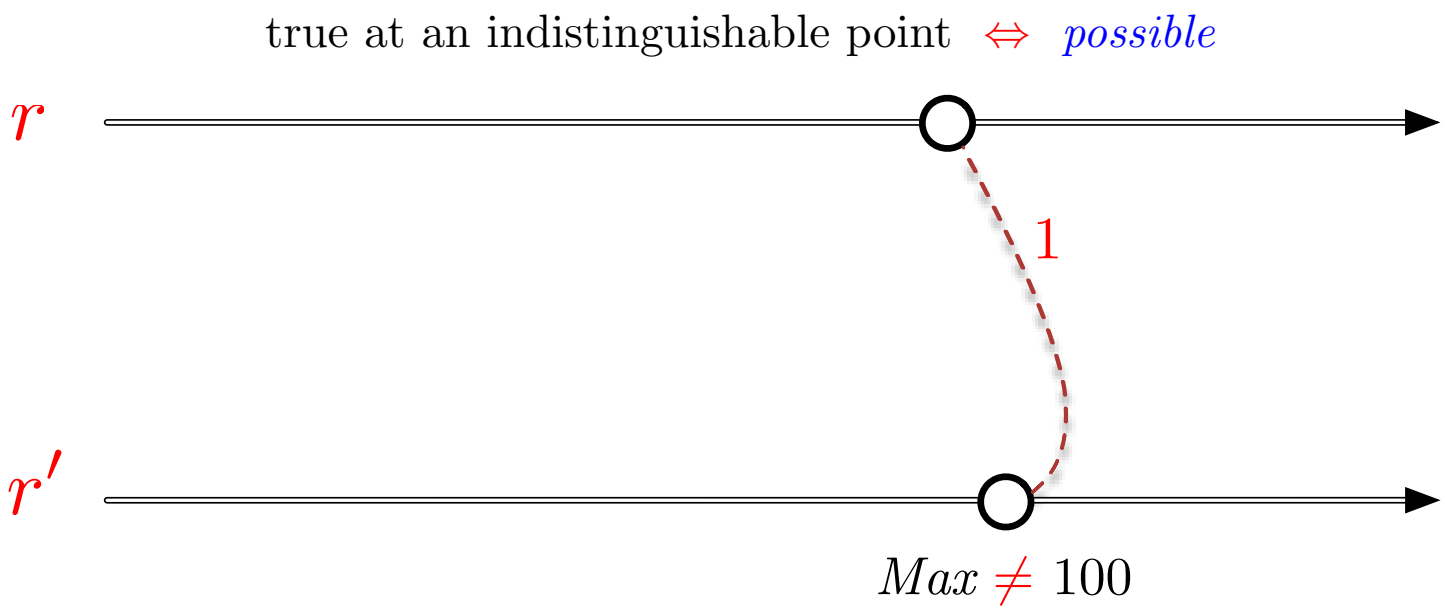
- Kripke 1950's, Hintikka [1962], Aumann [1976]
- Halpern and M. [1984]
- Parikh and Ramanujam [1985]
- Chandy and Misra [1986]
- Fagin *et al.* [1995], Reasoning about Knowledge

Basic notion: Indistinguishability

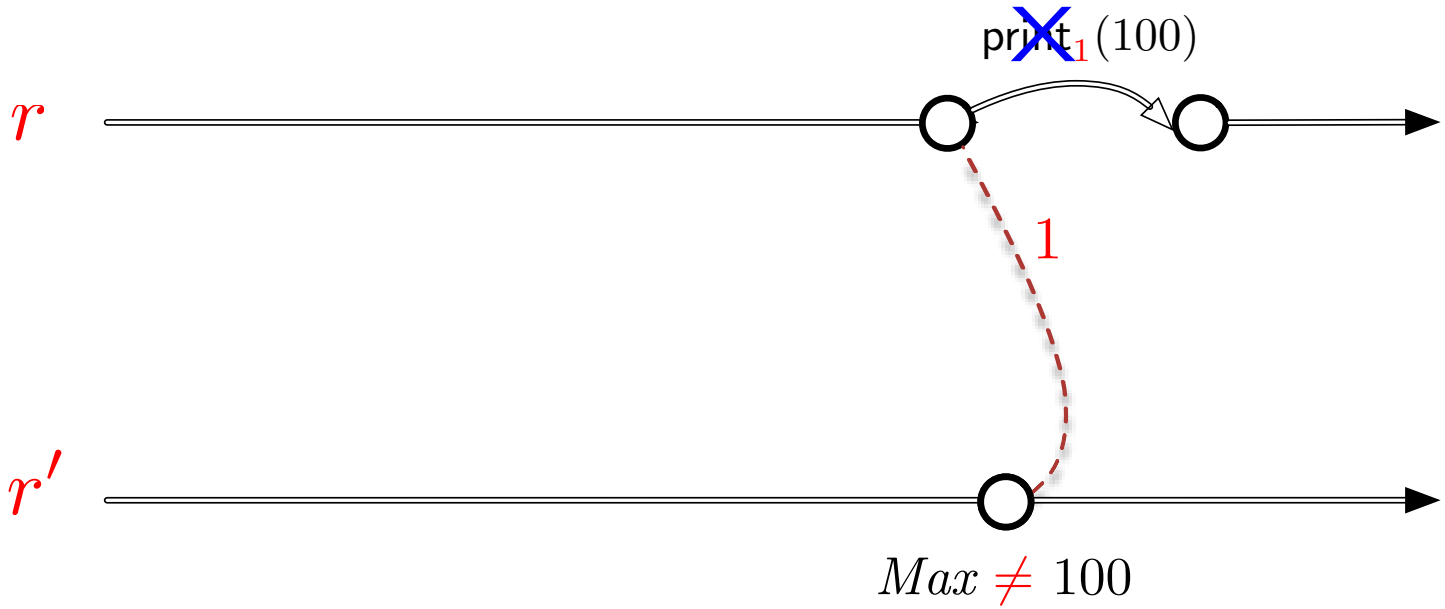


i has the same state at both points

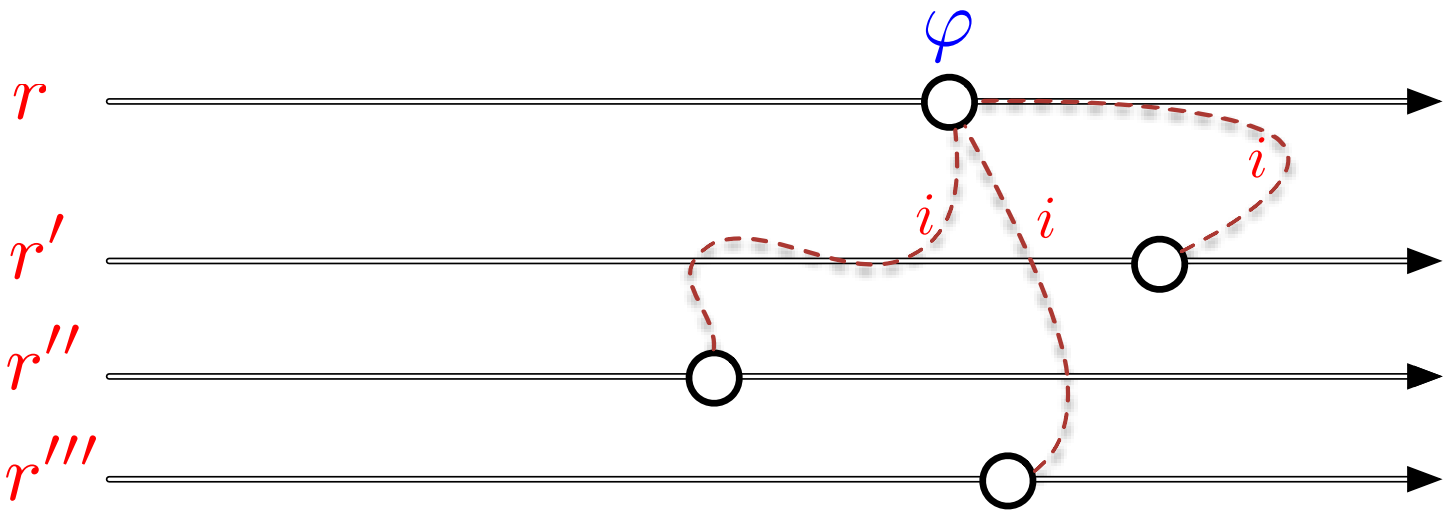
Basic notion: Indistinguishability



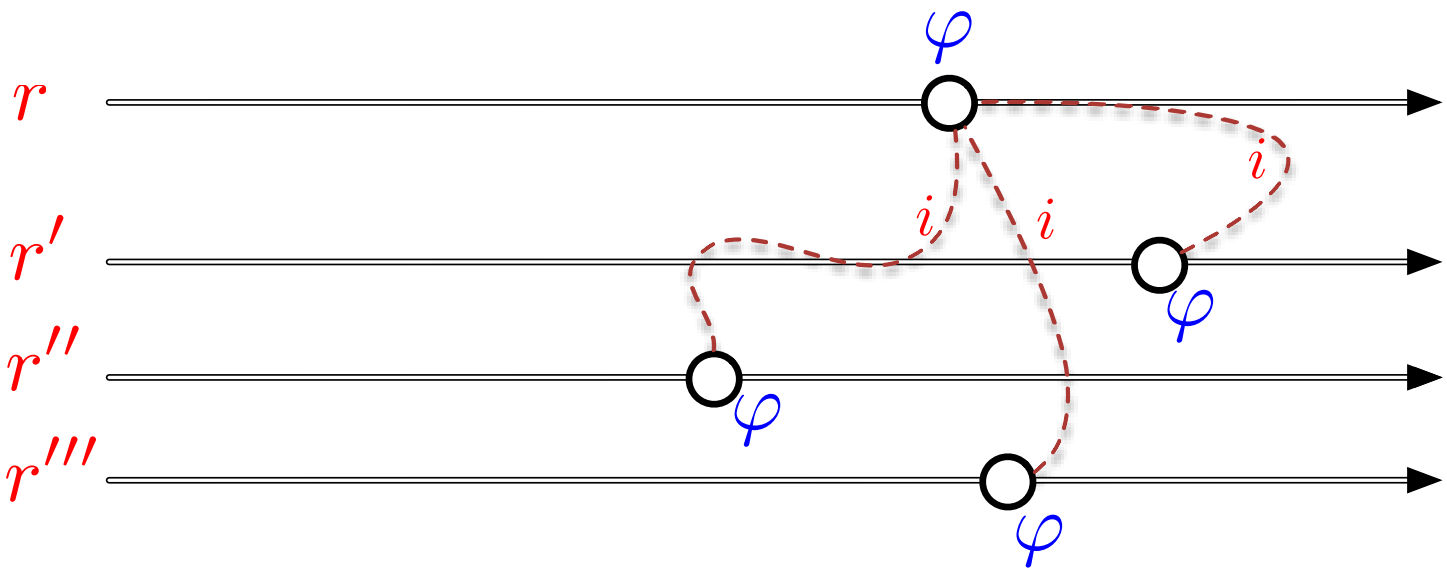
Basic notion: Indistinguishability



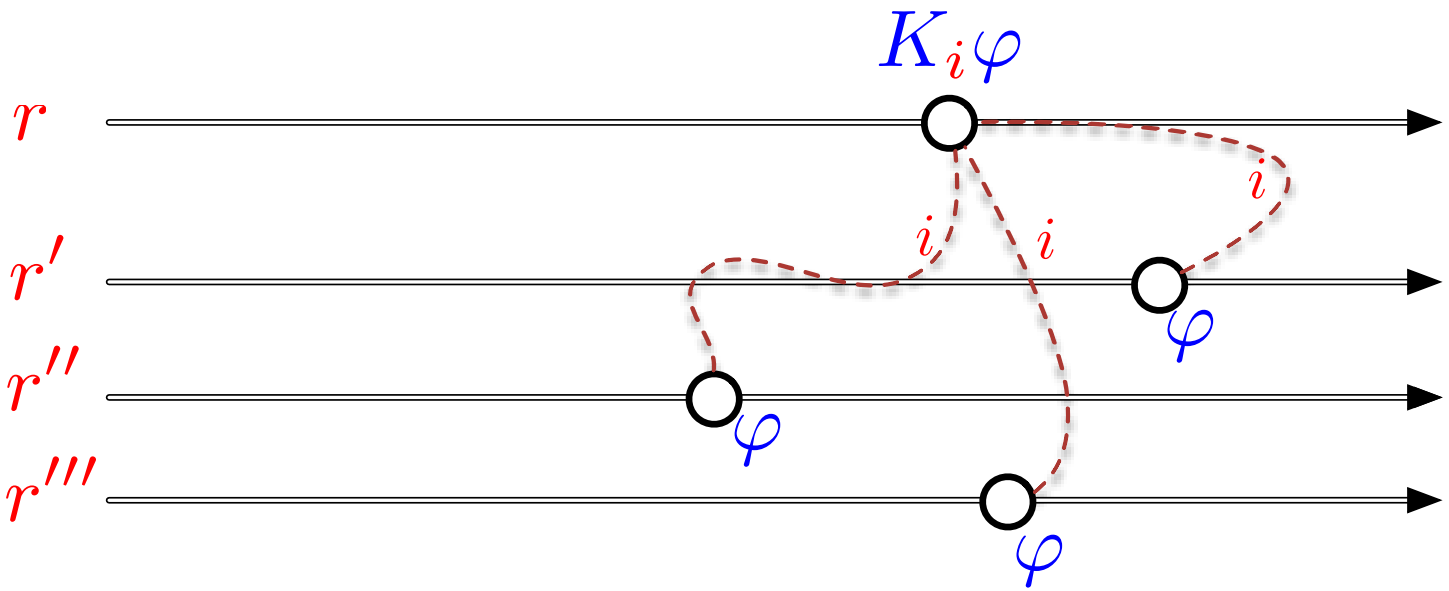
Defining Knowledge in Pictures



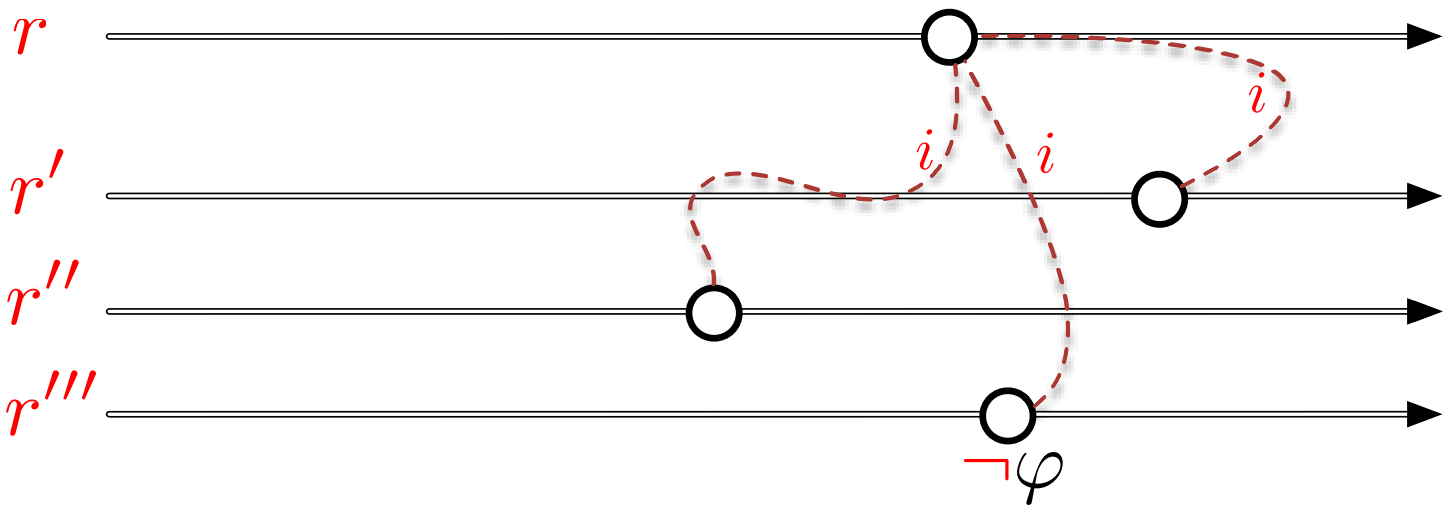
Defining Knowledge in Pictures



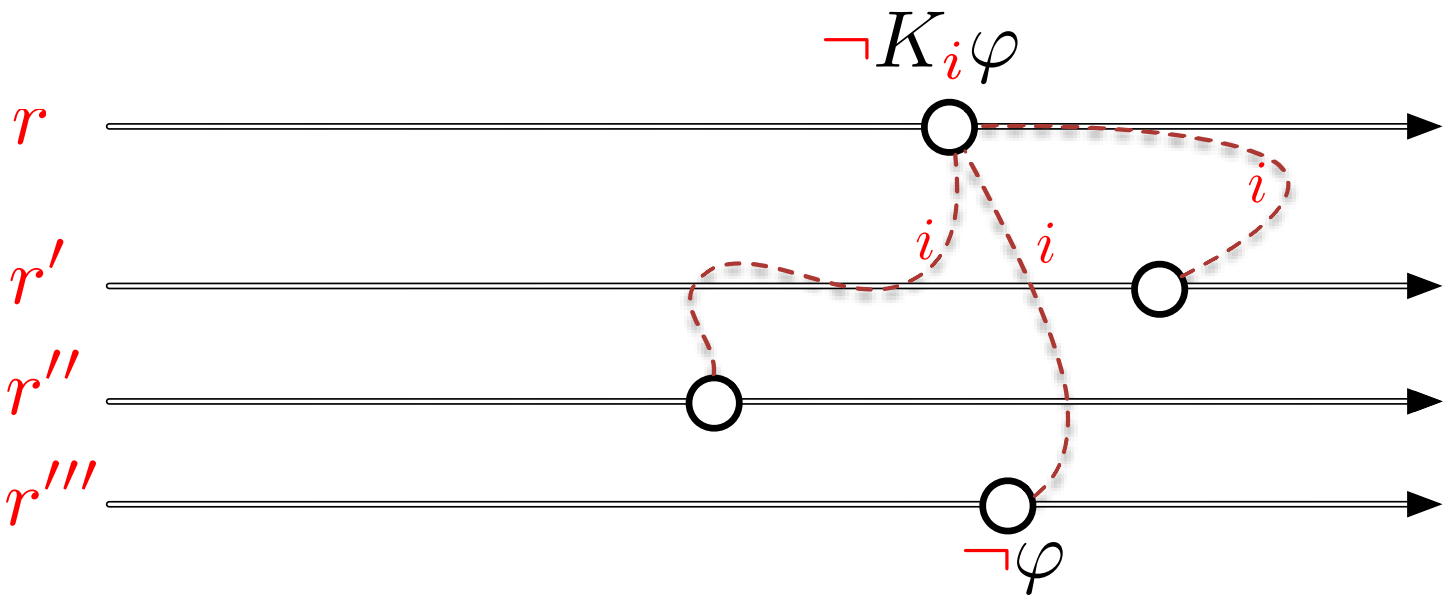
Defining Knowledge in Pictures



Defining Knowledge in Pictures



Defining Knowledge in Pictures



Defining Knowledge more formally [Fagin *et al.* 1995]

- A run is a sequence $r : \mathbb{N} \to \mathcal{G}$ of global states.
- A system is a set R of runs.

Assumption

Each global state $r(t)$ determines a *local state* $r_i(t)$ for every agent i.

Definition

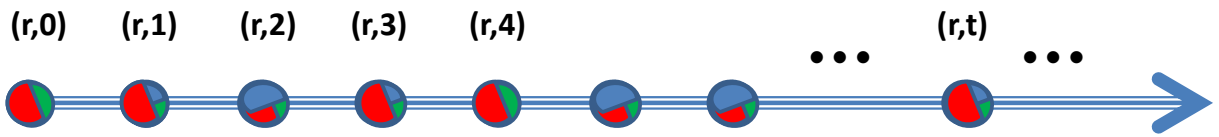
$$(R, r, t) \models K_i\varphi \quad \text{iff} \quad (R, r', t') \models \varphi \text{ for all points } (r', t') \text{ of } R \text{ such that } r_i(t) = r'_i(t').$$

Runs and points

A point (r, t) refers to time t in run r.

Facts are "true" or "false" at a point.

$R \times \mathbb{N} = \mathrm{Pts}(R)$ is the set of points in system R.



A Propositional Logic of Knowledge

Starting from a set Φ of primitive propositions, define $\mathcal{L}_n^K = \mathcal{L}_n^K(\Phi)$ by

$$\varphi \quad := \quad p \in \Phi \mid \neg\varphi \mid \varphi \wedge \varphi \mid K_1\varphi \mid \cdots \mid K_n\varphi$$

Given an interpretation $\pi : \Phi \times \mathrm{Pts}(R) \to \{\mathrm{True}, \mathrm{False}\}$

$(R, r, t) \models p$, for $p \in \Phi$, iff $\pi(p, r, t) = \mathrm{True}$.

$(R, r, t) \models \neg\varphi$ iff $(R, r, t) \not\models \varphi$

$(R, r, t) \models \varphi \wedge \psi$ iff both $(R, r, t) \models \varphi$ and $(R, r, t) \models \psi$.

Knowledge = Truth in All Possible Worlds

$$(\boldsymbol{R}, \boldsymbol{r}, \boldsymbol{t}) \models \boldsymbol{K_i}\varphi \quad \text{iff} \quad (R, r', t') \models \varphi \text{ for all points } (r', t') \text{ of } R \text{ such that } r_i(t) = r_i'(t').$$

Comments:

The definition ignores the complexity of computing knowledge

Local information = current local state.

$K_i\varphi$ holds if φ is guaranteed to hold in R given i's local state.

The knowledge operator K_i is an ***S5*** modal operator.

Knowledge = Truth in All Possible Worlds

$$(\boldsymbol{R}, \boldsymbol{r}, \boldsymbol{t}) \models \boldsymbol{K_i}\varphi \quad \text{iff} \quad (R, r', t') \models \varphi \text{ for all points } (r', t') \text{ of } R \text{ such that } r_i(t) = r_i'(t').$$

Comments:

The definition ignores the complexity of computing knowledge

Local information = current local state.

$K_i\varphi$ holds if φ is guaranteed to hold in R given i's local state.

The knowledge operator K_i is an ***S5*** modal operator.

Necessary Conditions for Actions (aka "preconditions")

$Max = c$ is a necessary condition for $\texttt{print}_1(c)$ in CTM.

Definition

ψ is a necessary condition for $\texttt{does}_i(\alpha)$ in R if

$$(R, r, t) \models \texttt{does}_i(\alpha) \Rightarrow \psi \quad \text{for all } (r, t) \in \text{Pts}(R).$$

Specifications impose necessary conditions:

- "Good credit" is necessary for ATM dispensing cash
- "CS is empty" is necessary for entering the CS in Mutual Exclusion

Knowledge of Preconditions

Definition

α is a ***conscious action*** for i in R if

$(R, r, t) \vDash \mathtt{does}_i(\alpha)$ & $r_i'(t') = r_i(t)$ implies $(R, r', t') \vDash \mathtt{does}_i(\alpha)$

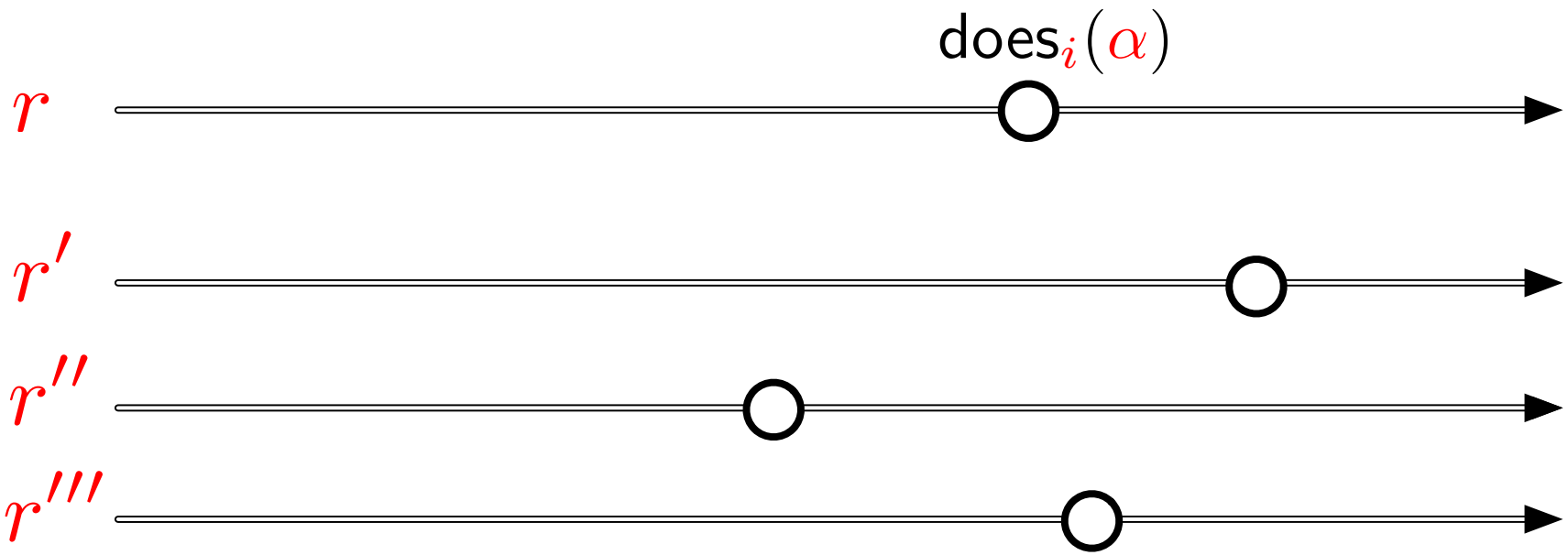
Theorem (KoP, [M. 2015])

Suppose that α is a conscious action for i in the system R.

If φ is a necessary condition for $\mathtt{does}_i(\alpha)$ in R, then

$K_i\varphi$ is a necessary condition for $\mathtt{does}_i(\alpha)$ in R.

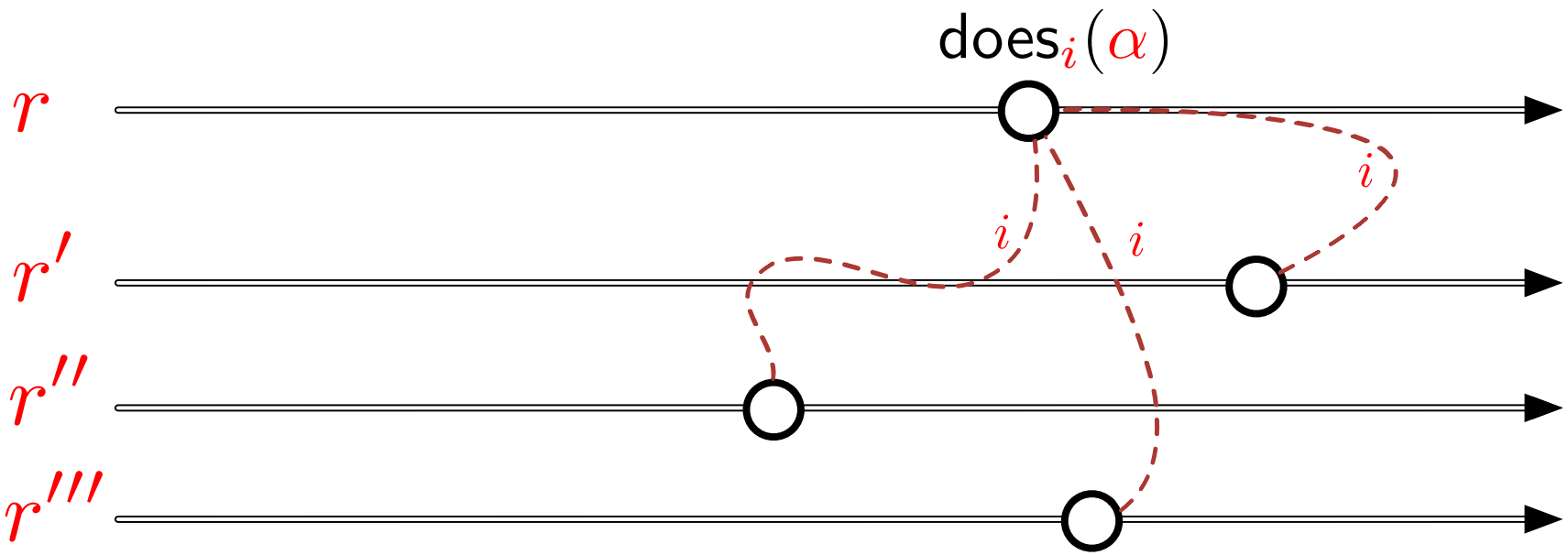
Proof of **KoP**



α is a conscious action for ***i***

φ is a necessary condition for $\mathtt{does}_i(\alpha)$

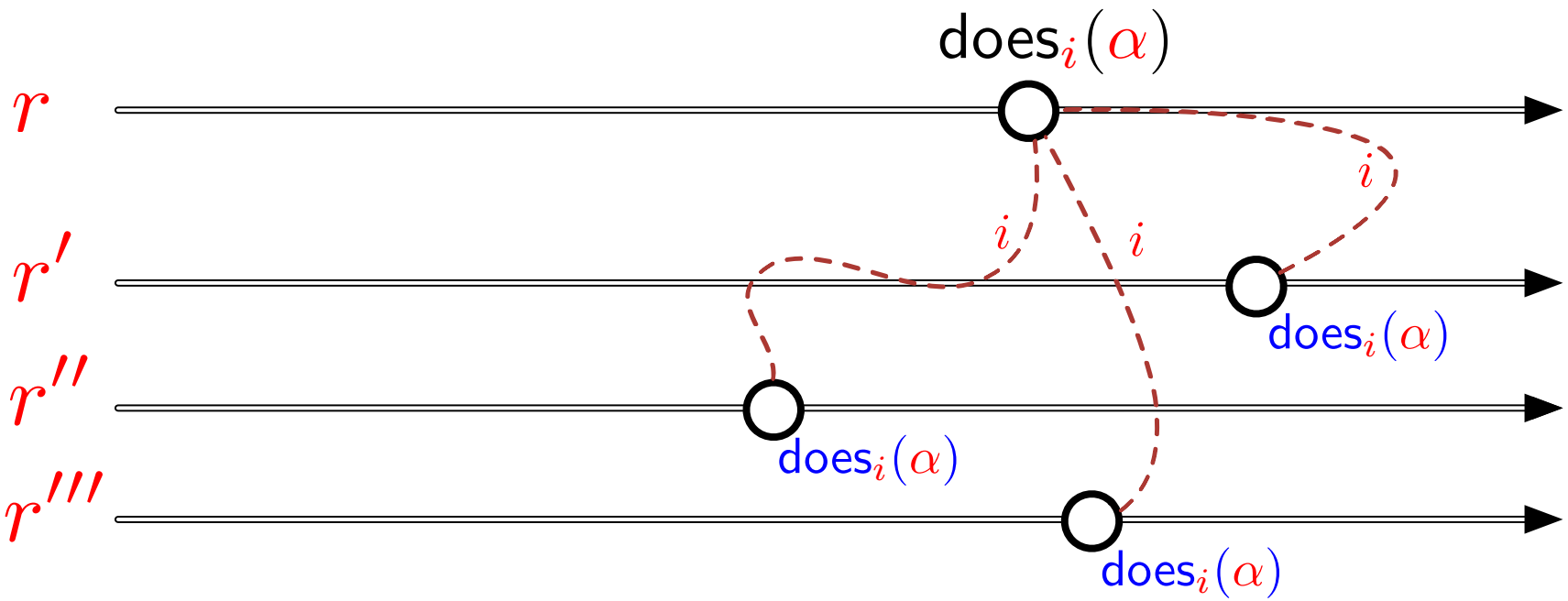
Proof of **KoP**



α is a conscious action for ***i***

φ is a necessary condition for $\mathtt{does}_i(\alpha)$

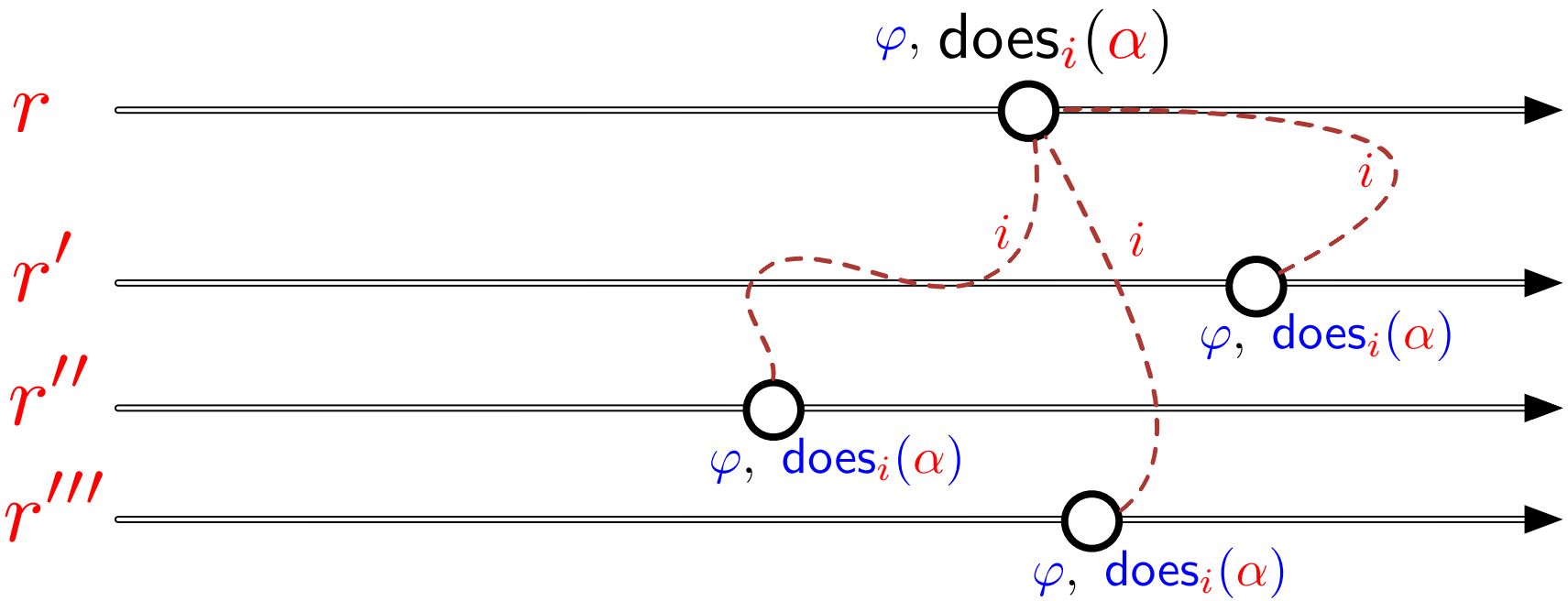
Proof of **KoP**



α is a conscious action for ***i***

φ is a necessary condition for $\mathtt{does}_i(\alpha)$

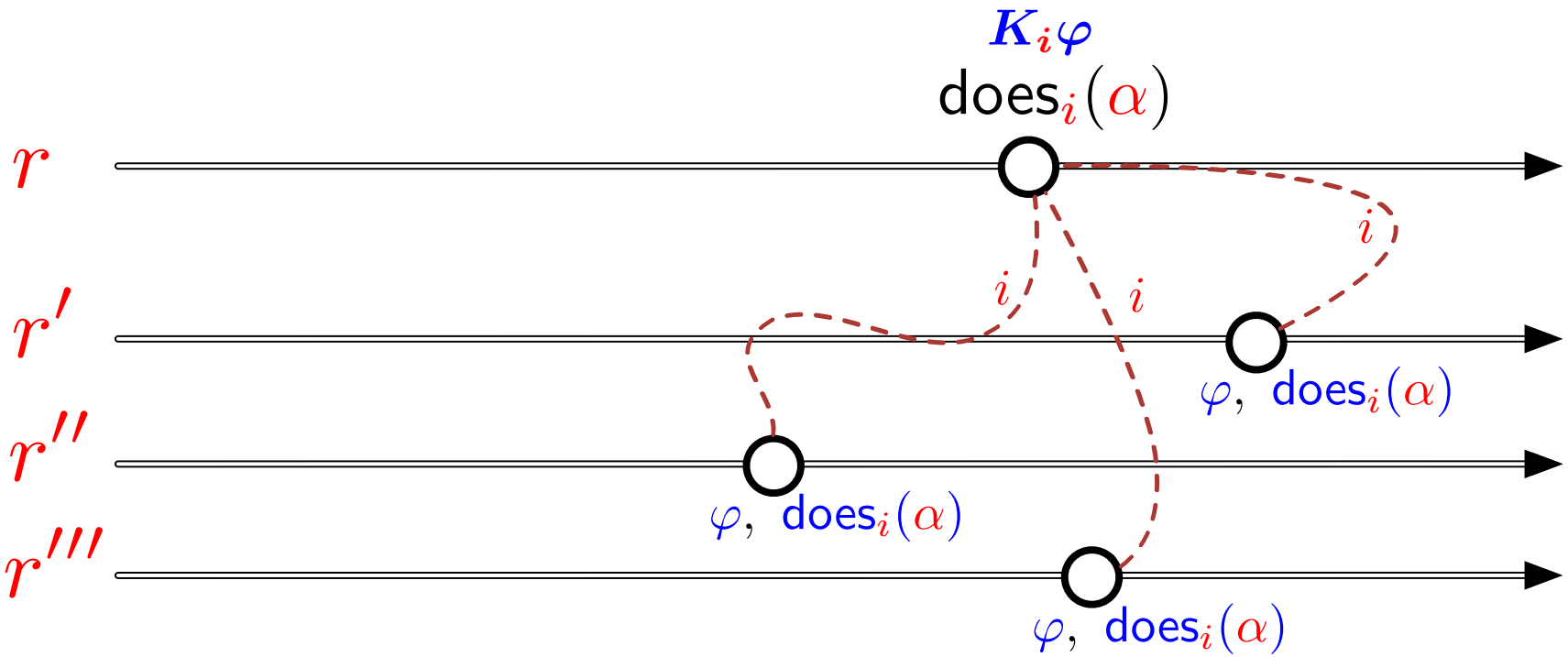
Proof of **KoP**



α is a conscious action for ***i***

φ is a necessary condition for $\mathtt{does}_i(\alpha)$

Proof of **KoP**

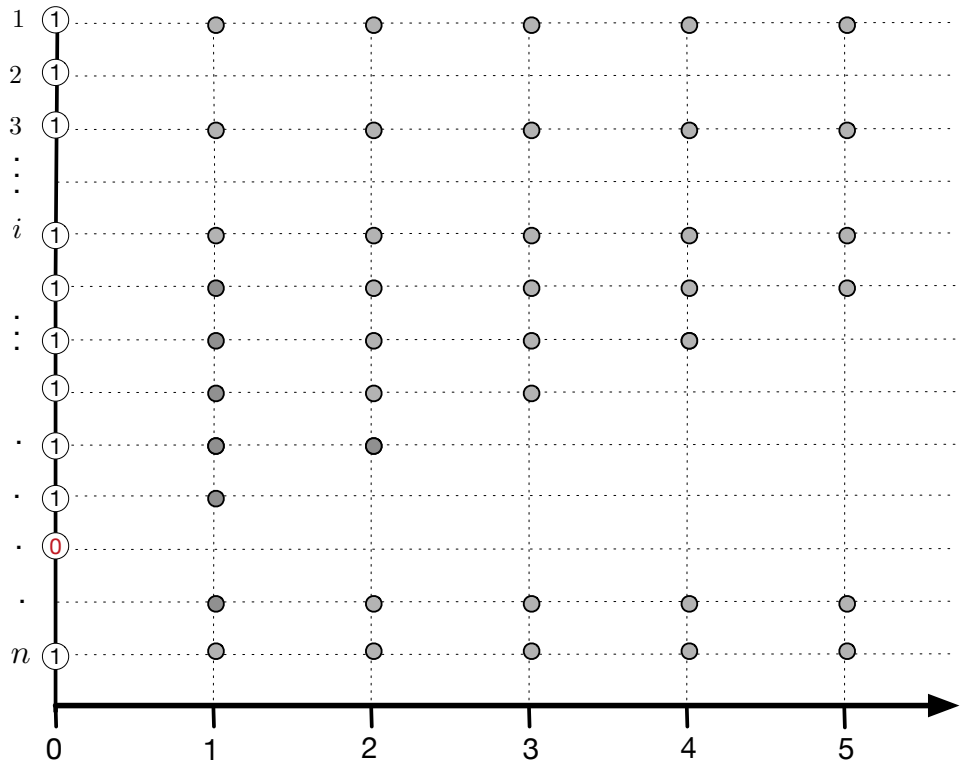


α is a conscious action for i

φ is a necessary condition for $\text{does}_i(\alpha)$

An Application: Distributed Consensus

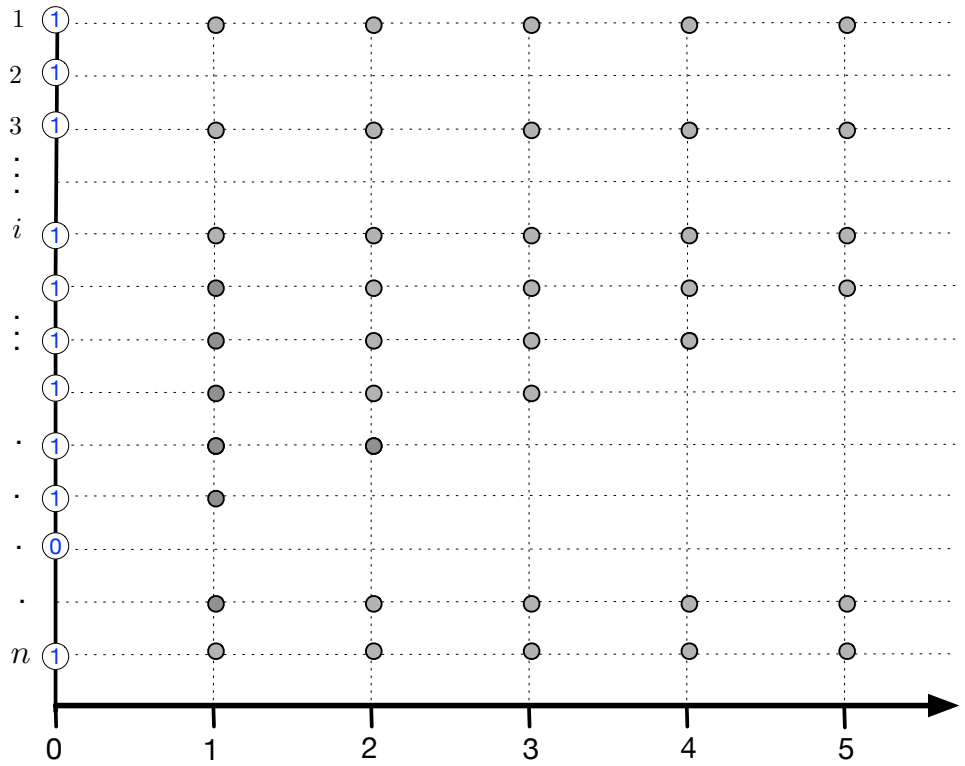
Model:



a complete communication graph with n nodes

An Application: Distributed Consensus

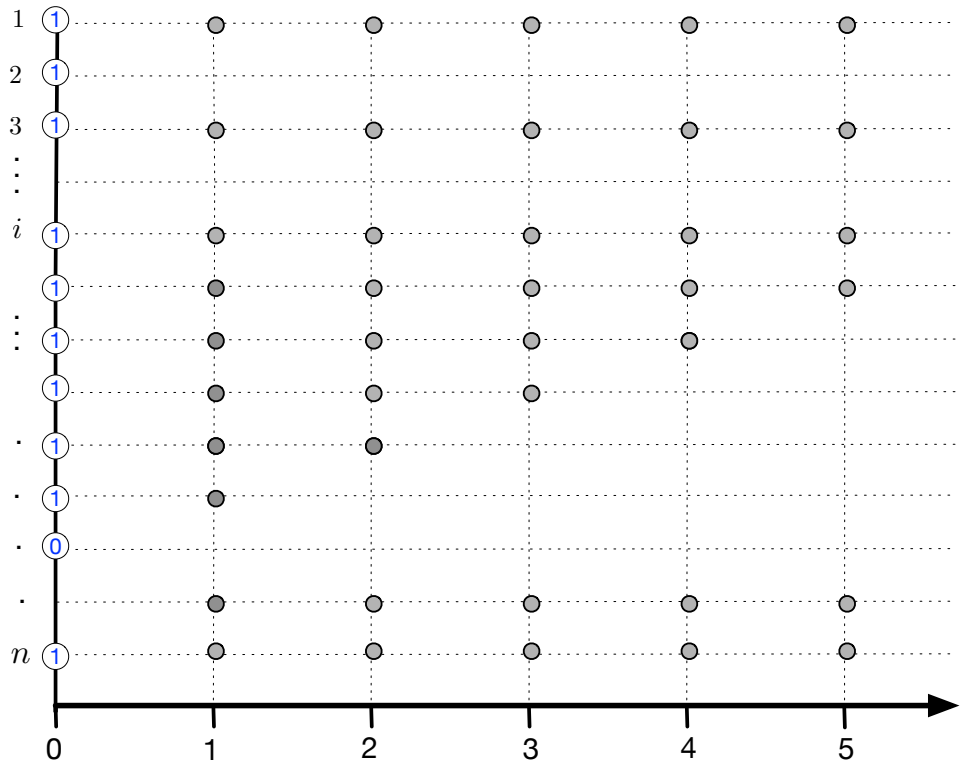
Model:



each process i starts with an initial "vote" $v_i \in \{0, 1\}$

An Application: Distributed Consensus

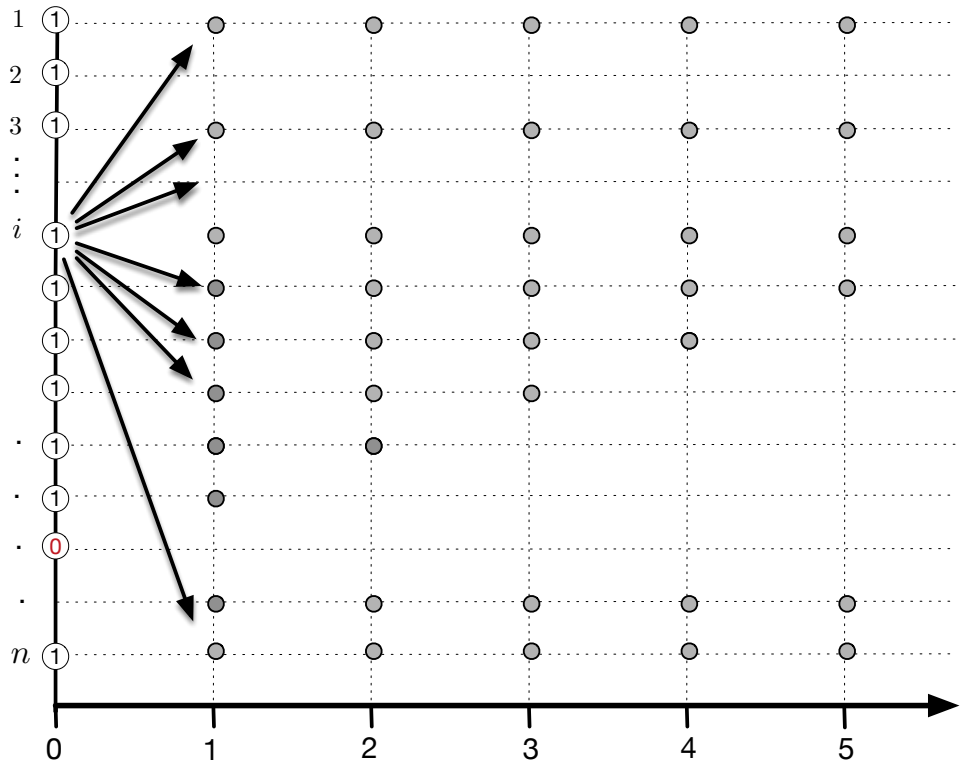
Model:



a discrete global clock, messages take 1 round

An Application: Distributed Consensus

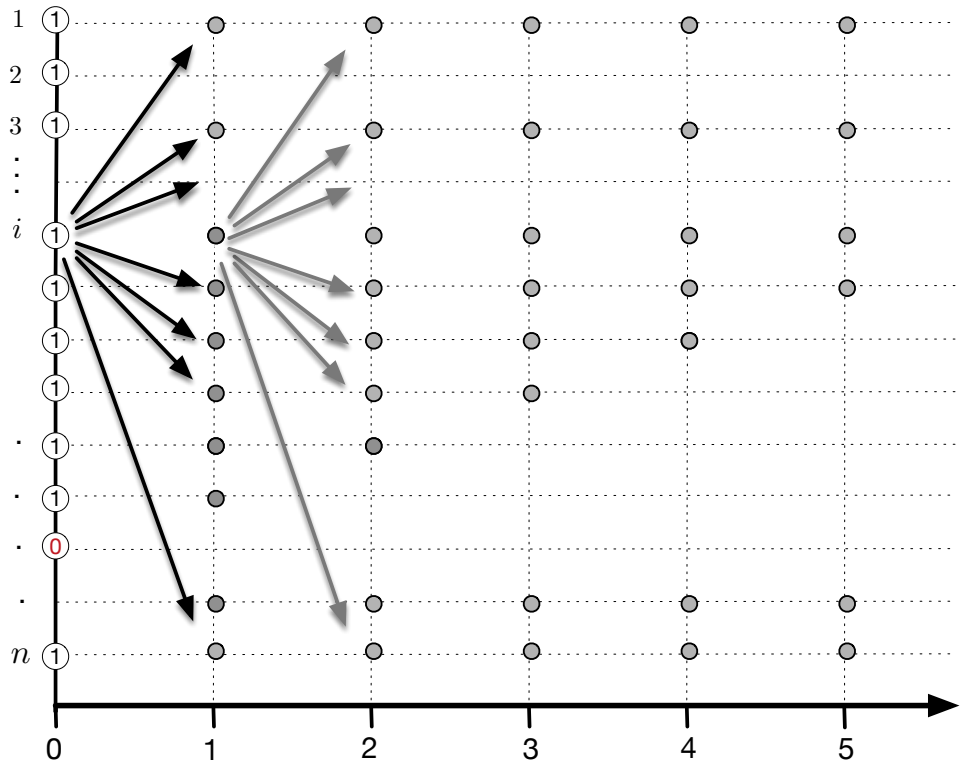
Model:



full-information protocol (fip): Processes broadcast their complete history

An Application: Distributed Consensus

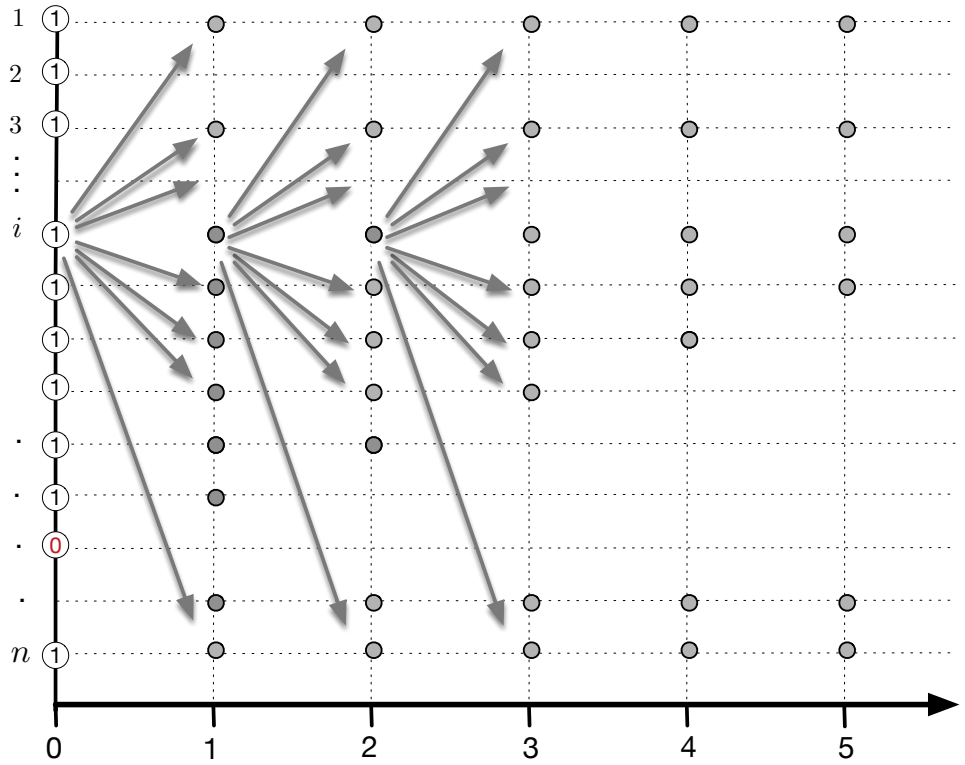
Model:



full-information protocol (fip): Processes broadcast their complete history

An Application: Distributed Consensus

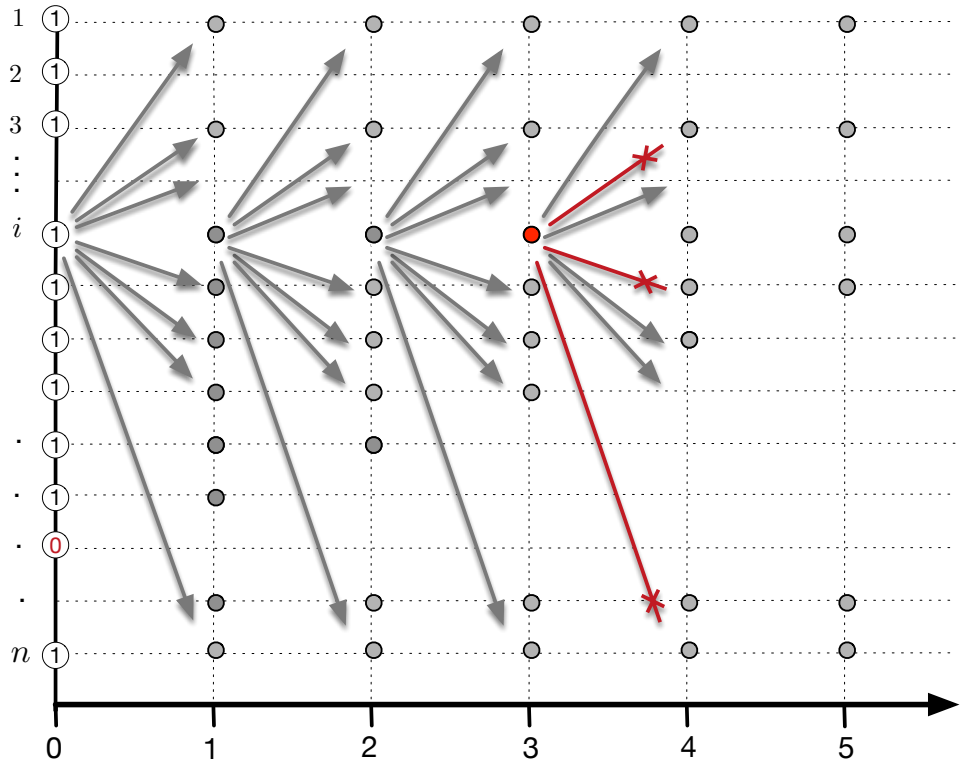
Model:



full-information protocol (fip): Processes broadcast their complete history

An Application: Distributed Consensus

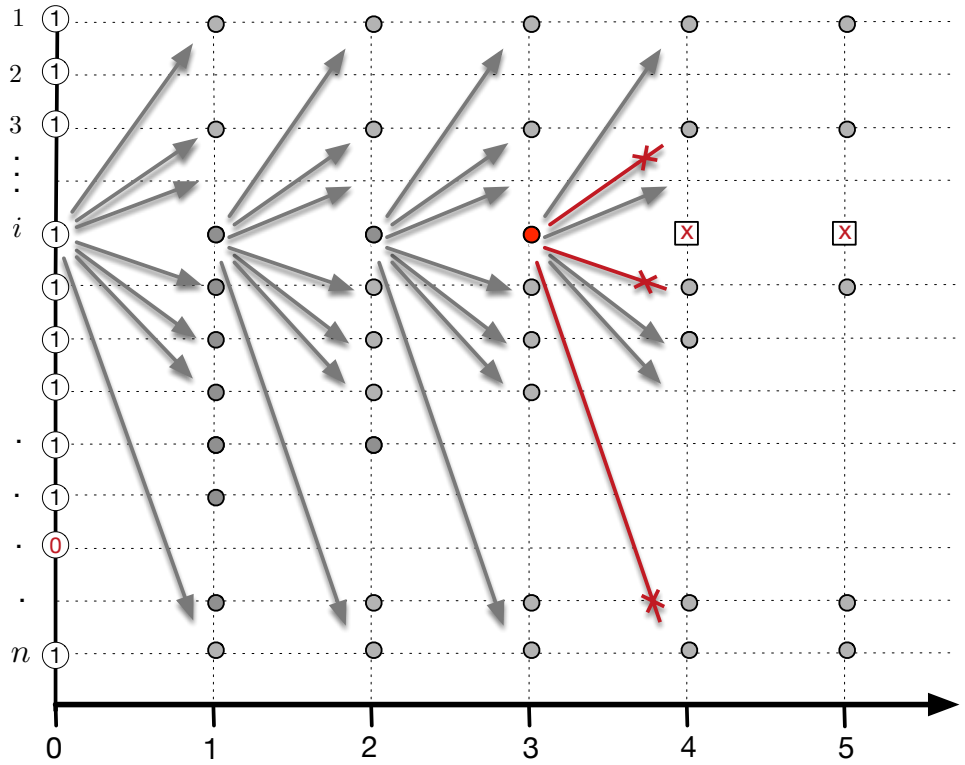
Model:



crash failures

An Application: Distributed Consensus

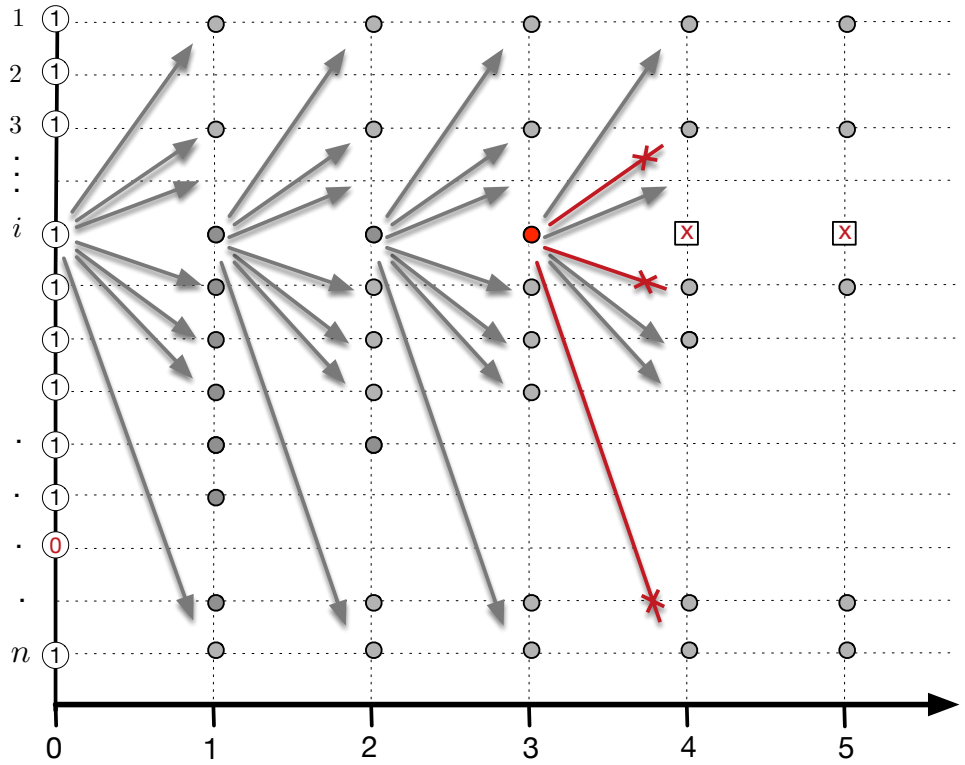
Model:



crash failures

An Application: Distributed Consensus

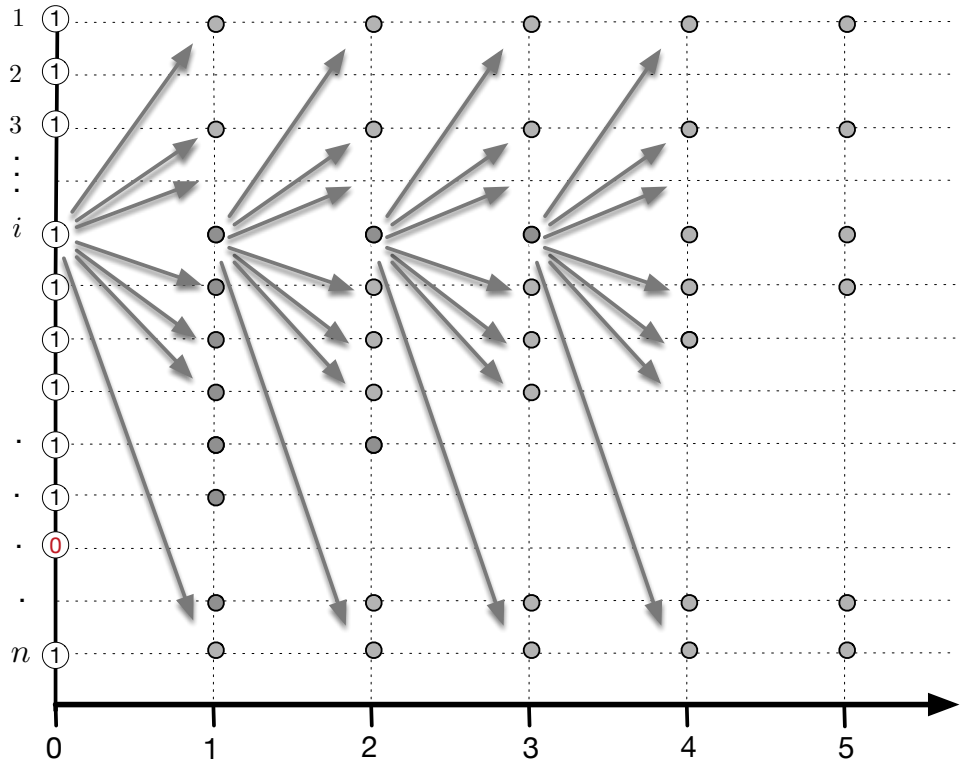
Model:



crash failures: at most $t < n$ processes fail per run

An Application: Distributed Consensus

Model:



a process is correct in r if it doesn't crash

Consensus

Protocol Specification:

In every run with no more than t processes crashes:

Decision: Every correct process must decide on some value

Validity: $\texttt{decide}_i(v)$ is allowed only if someone voted v ("$\exists v$")

Agreement: All correct processes decide on the same value

correct = does not crash

$t+1$ round Lower Bound

Theorem (Dolev-Strong '82, Fischer-Lynch '82)

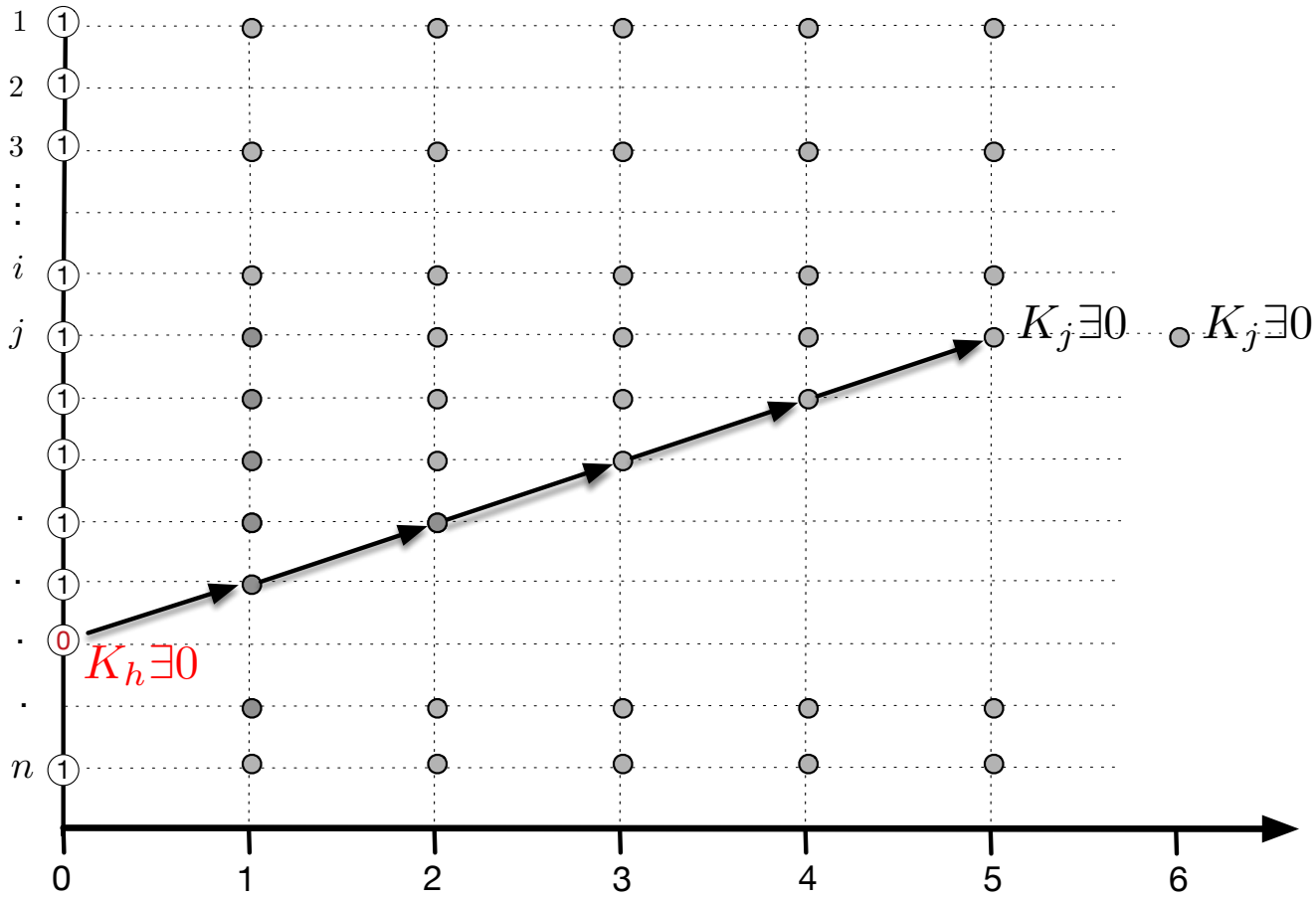
Every consensus protocol must have a (worst-case) run in which the last correct process requires at least $t+1$ rounds to decide.

A Knowledge-based Analysis: Deciding on 0

Validity: A necessary condition for $\texttt{decide}_i(v)$ is $\exists v$, for $v = 0, 1$

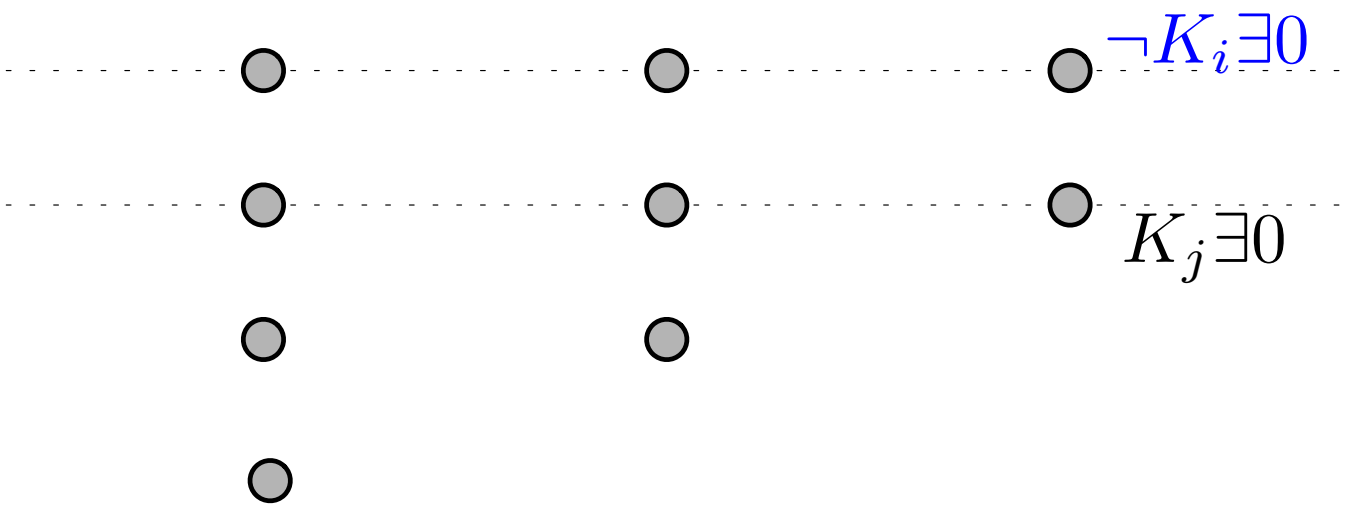
- By **Validity**, $\exists 0$ is a necessary condition for $\texttt{decide}_i(0)$.
- By **KoP**, $K_i \exists 0$ is a necessary condition for $\texttt{decide}_i(0)$.

Knowing ∃0



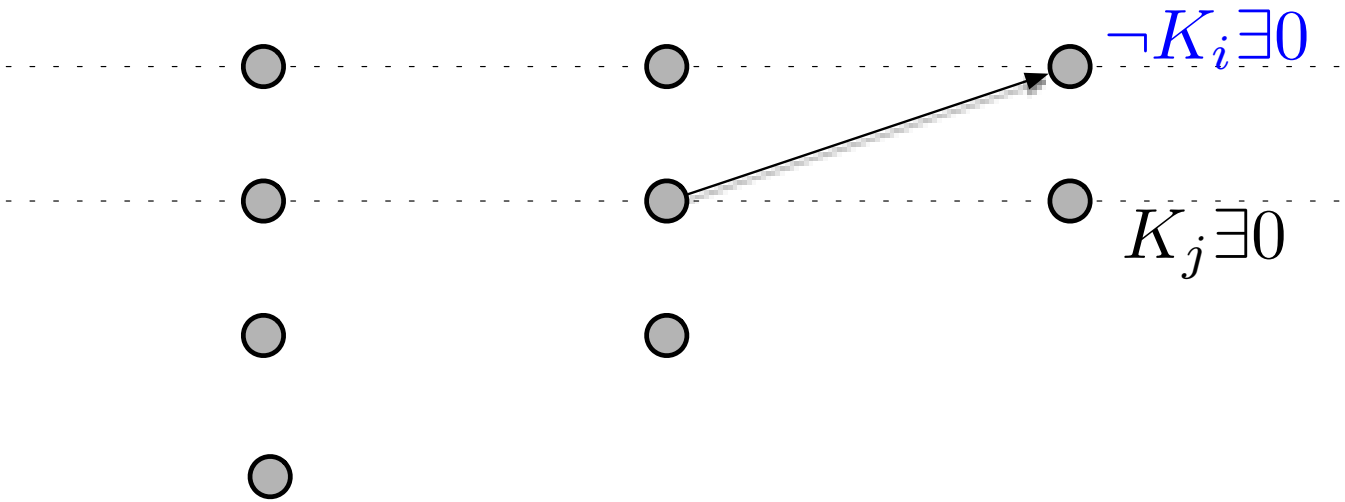
$K_j\exists 0$ holds if $v_j = 0$ or j received a message from a process that knows $\exists 0$.

Knowing $\exists 0$



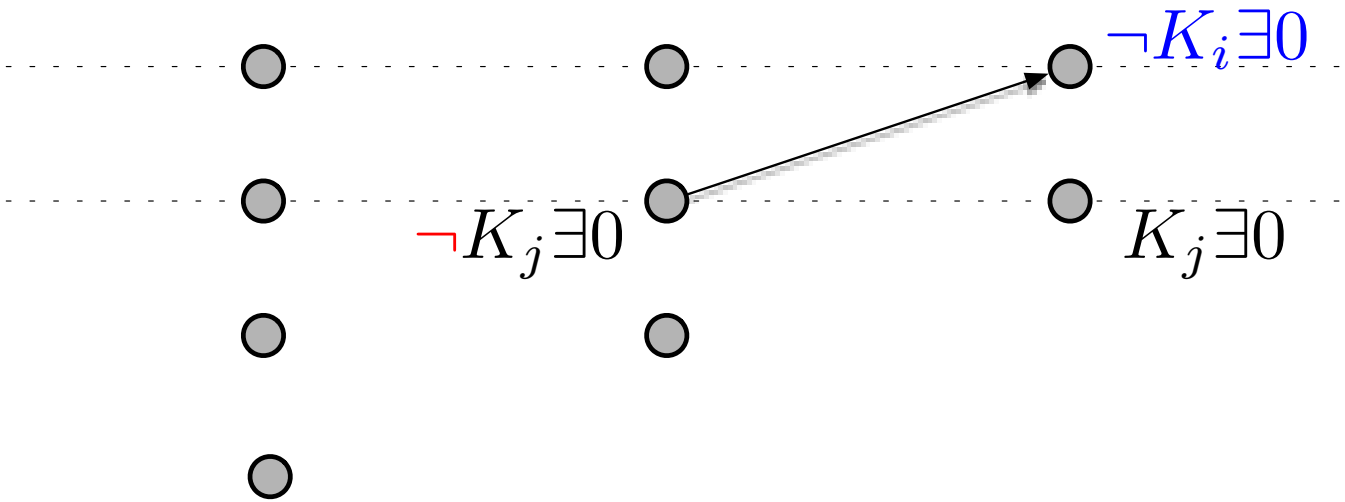
How can one proc know $\exists 0$ when another does not?

Knowing $\exists 0$



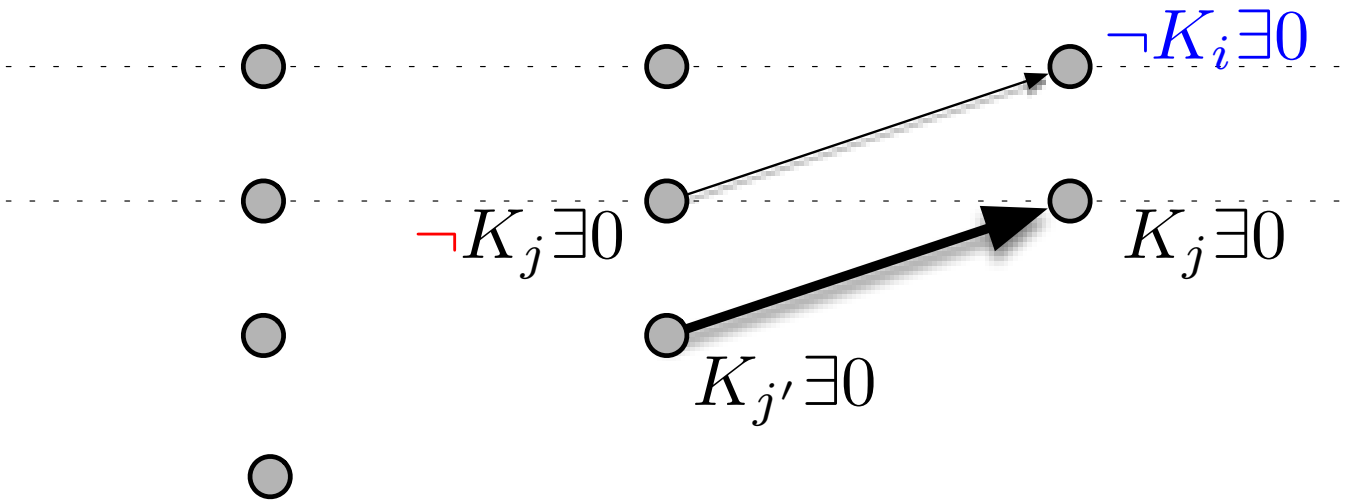
How can one proc know $\exists 0$ when another does not?

Knowing $\exists 0$



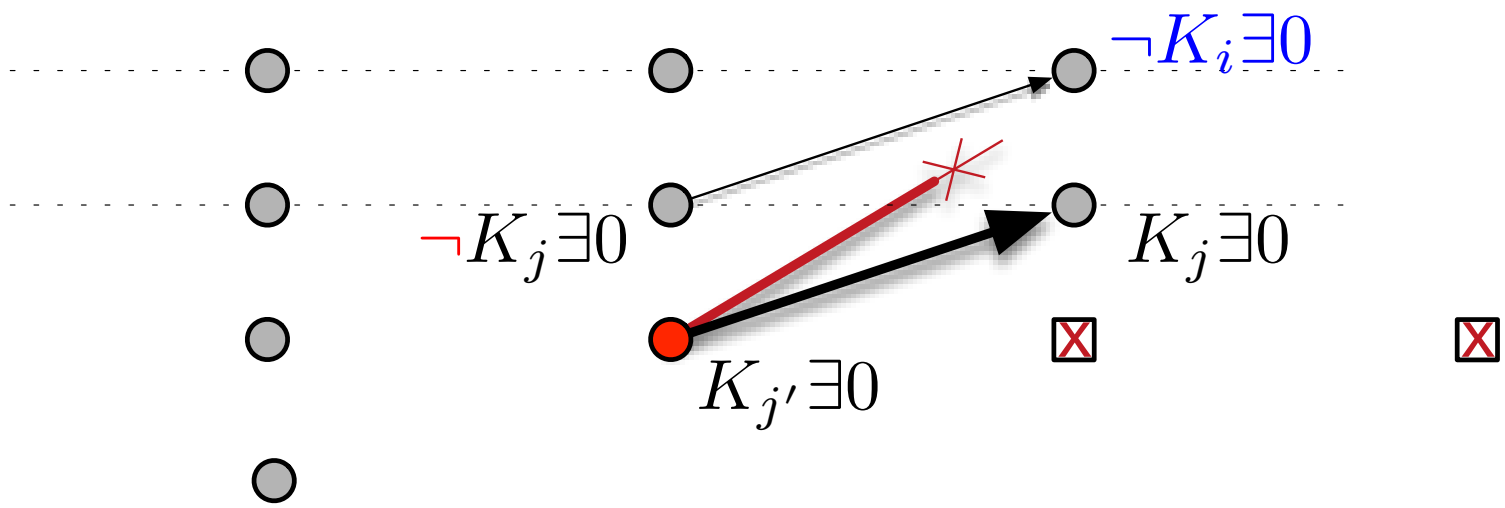
How can one proc know $\exists 0$ when another does not?

Knowing $\exists 0$



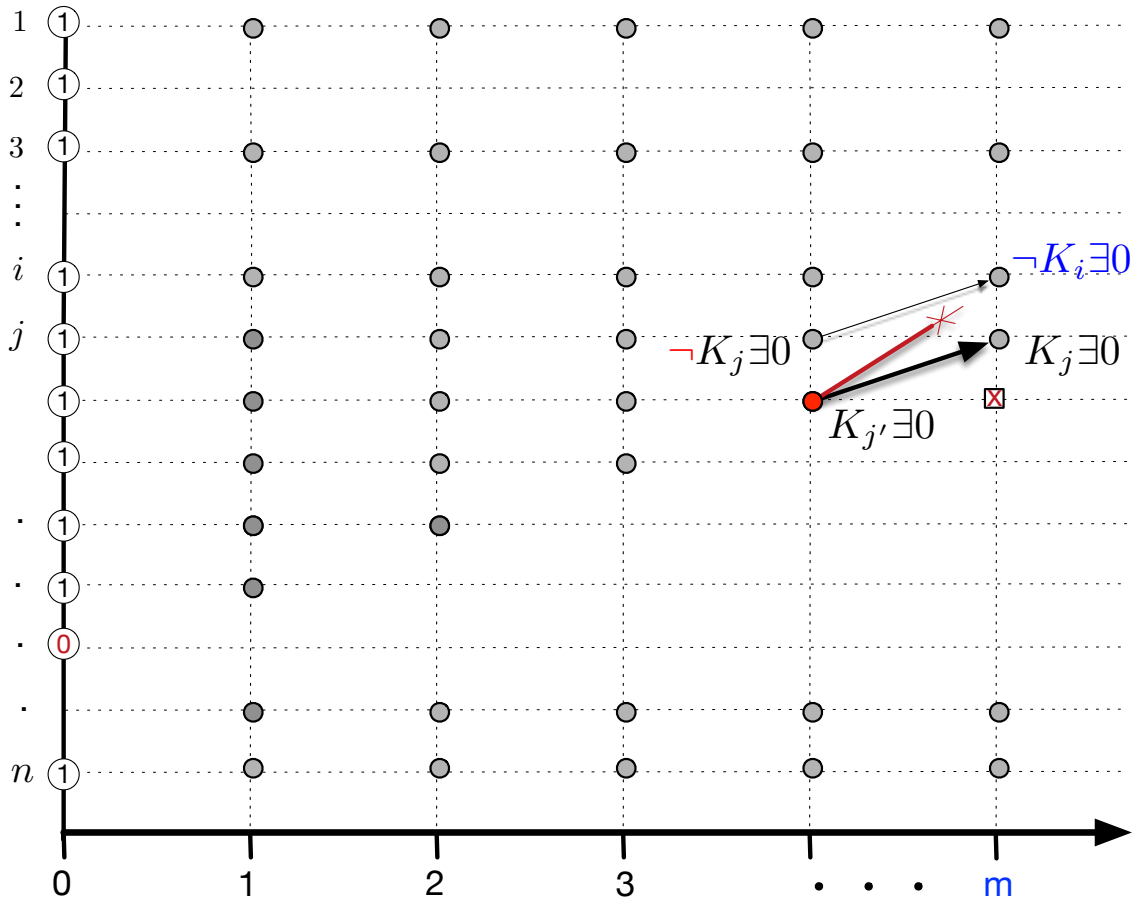
How can one proc know $\exists 0$ when another does not?

Knowing ∃0

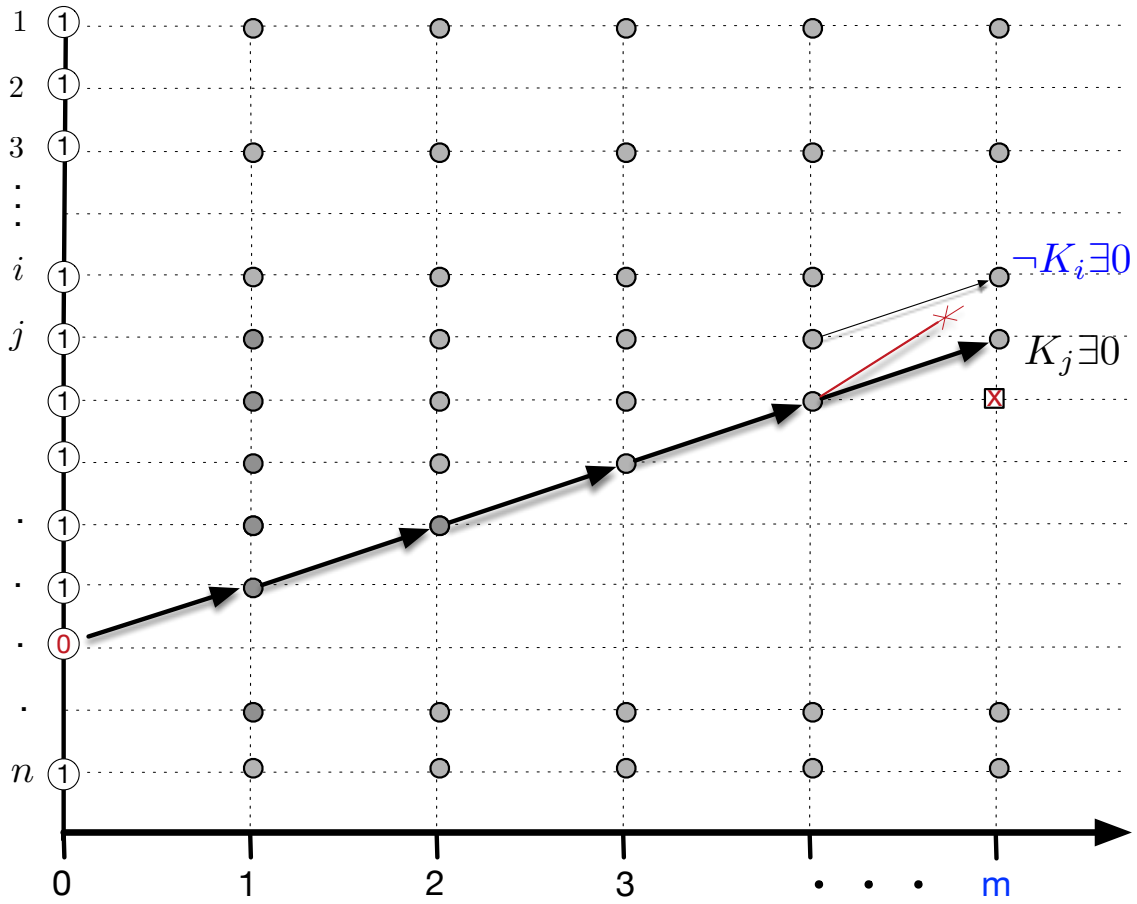


How can one proc know ∃0 when another does not?

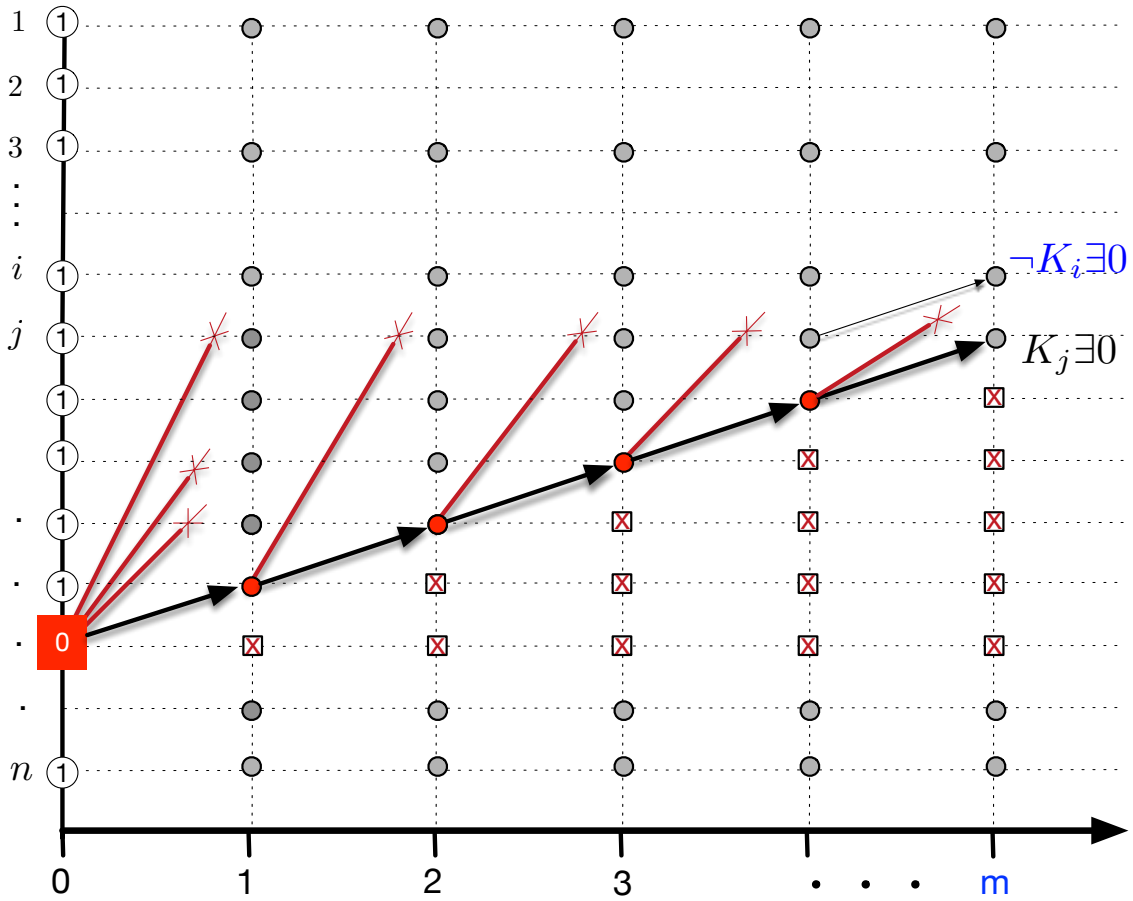
Knowing ∃0



Knowing ∃0

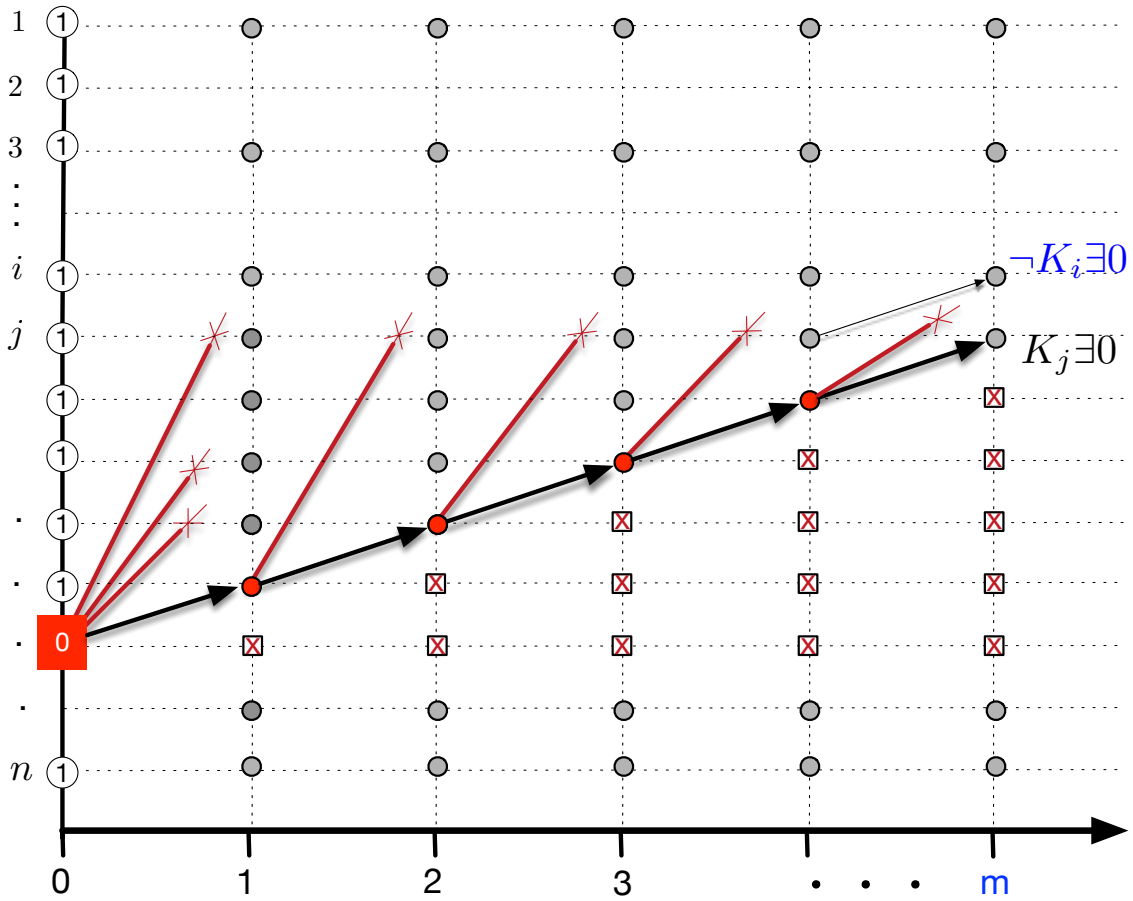


Knowing $\exists 0$



Claim: If $K_j \exists 0$ & $\neg K_i \exists 0$ at time m, then $\geq m$ crashes have occurred

Knowing ∃0



Corollary: At time $t+1$, either everyone knows ∃0 or nobody does

A Simple Consensus Protocol

Protocol P_0 (for undecided process i):

if $\texttt{time} = t+1$ & $K_i \exists 0$ **then** $\texttt{decide}_i(0)$

elseif $\texttt{time} = t+1$ & $\neg K_i \exists 0$ **then** $\texttt{decide}_i(1)$

Communication is according to the fip.

Optimal: All decisions at time $t+1$

A Better Protocol

Protocol Q_0 (for undecided process i):

if $K_i \exists 0$ **then** $\texttt{decide}_i(0)$

elseif $\texttt{time} = t+1$ & $\neg K_i \exists 0$ **then** $\texttt{decide}_i(1)$

Optimal: All decisions by time $t+1$

Q_0 Dominates P_0

An adversary is a pair $\beta = (V, F)$

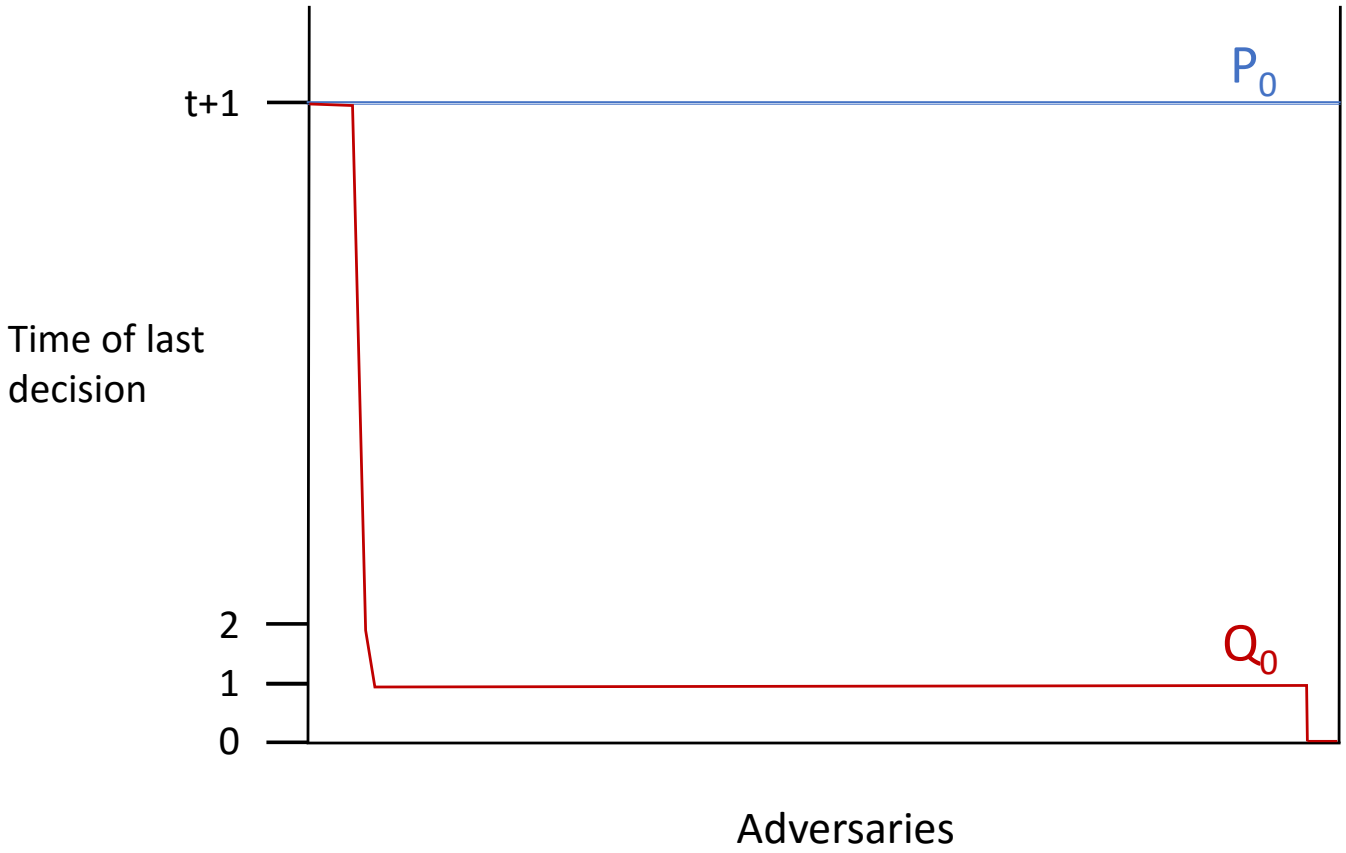
- $V = (v_1, \ldots, v_n)$ determines the initial values
- F is the failure pattern — who crashes, when, and how

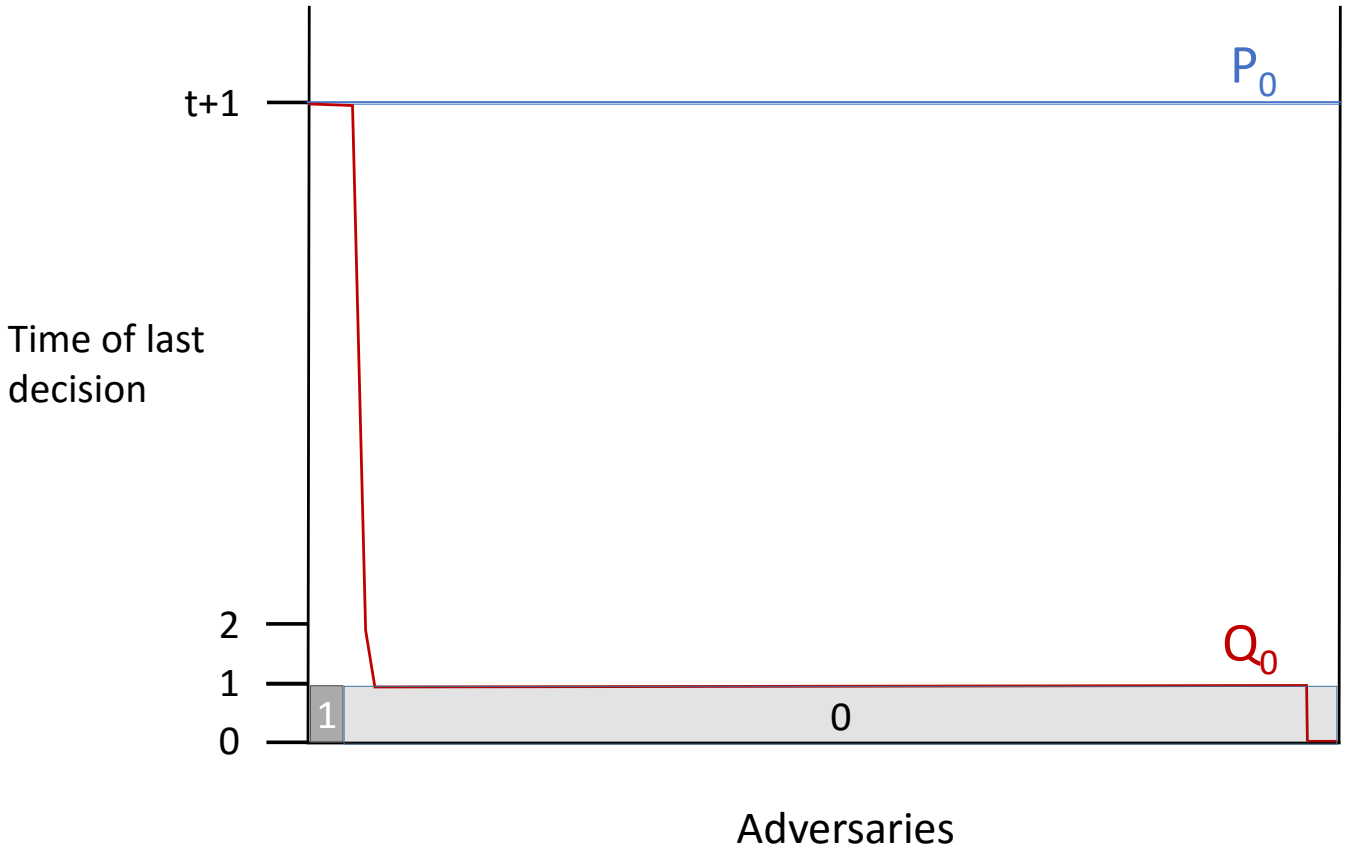
Definition

Protocol P' dominates protocol P if, for all adv. β, process i and time k, if i decides at time k in $P[\beta]$ then it decides by time k in $P'[\beta]$.

Claim

Q_0 strictly dominates P_0.


t+1
2
1
0
Time of last decision
P_0
Q_0
Adversaries


t+1
Time of last decision
2
1
0
P_0
Q_0
1
0
Adversaries

Q_0 Dominates P_0

An adversary is a pair $\beta = (V, F)$

- $V = (v_1, \ldots, v_n)$ determines the initial values
- F is the failure pattern — who crashes, when, and how

Definition

Protocol P' dominates protocol P if, for all adv. β, process i and time k, if i decides at time k in $P[\beta]$ then it decides by time k in $P'[\beta]$.

Claim

Q_0 strictly dominates P_0.

Can Q_0 be dominated?

A Knowledge-based Analysis: Deciding on 1

Suppose the rule for deciding 0 is $K_j \exists 0 \Leftrightarrow \texttt{decide}_j(0)$.

When can $\texttt{decide}_i(1)$ be performed?

Recall:

> **Agreement:** All correct processes decide on the same value

- By Agreement, "no currently active process decides 0" is a necessary condition for $\texttt{decide}_i(1)$; so
- ψ = "no active process knows $\exists 0$" is a nec. cond. for $\texttt{decide}_i(1)$;
- By the **KoP**, $K_i\big(\texttt{nobody\_knows} \exists 0\big)$ is a necessary condition for $\texttt{decide}_i(1)$.

An Unbeatable Consensus Protocol

[Castañeda, Gonczarowski & M. '14]

Protocol OPT_0 (for undecided process i):

if	$K_i \exists 0$	**then** $\texttt{decide}_i(0)$
elseif	$K_i(\texttt{nobody\_knows} \exists 0)$	**then** $\texttt{decide}_i(1)$

By the **K**_o**P**:

- $\texttt{decide}_i(0)$ is performed as soon as possible
- $\texttt{decide}_i(1)$ is performed asap, given the rule for $\texttt{decide}(0)$

An Unbeatable Consensus Protocol

[Castañeda, Gonczarowski & M. '14]

Protocol OPT_0 (for undecided process i):

if	$K_i \exists 0$	**then** $\texttt{decide}_i(0)$
elseif	$K_i(\texttt{nobody\_knows} \exists 0)$	**then** $\texttt{decide}_i(1)$

By the **K***o***P**:

- $\texttt{decide}_i(0)$ is performed as soon as possible
- $\texttt{decide}_i(1)$ is performed asap, given the rule for $\texttt{decide}(0)$

Implemented OPT_0

[Castañeda, Gonczarowski & M. '14]

Protocol OPT_0 (for undecided process i):

if $K_i \exists 0$ **then** $\texttt{decide}_i(0)$

elseif $K_i(\texttt{nobody\_knows} \exists 0)$ **then** $\texttt{decide}_i(1)$

My name is Sherlock Holmes.
It is my business to know
what other people don't know.

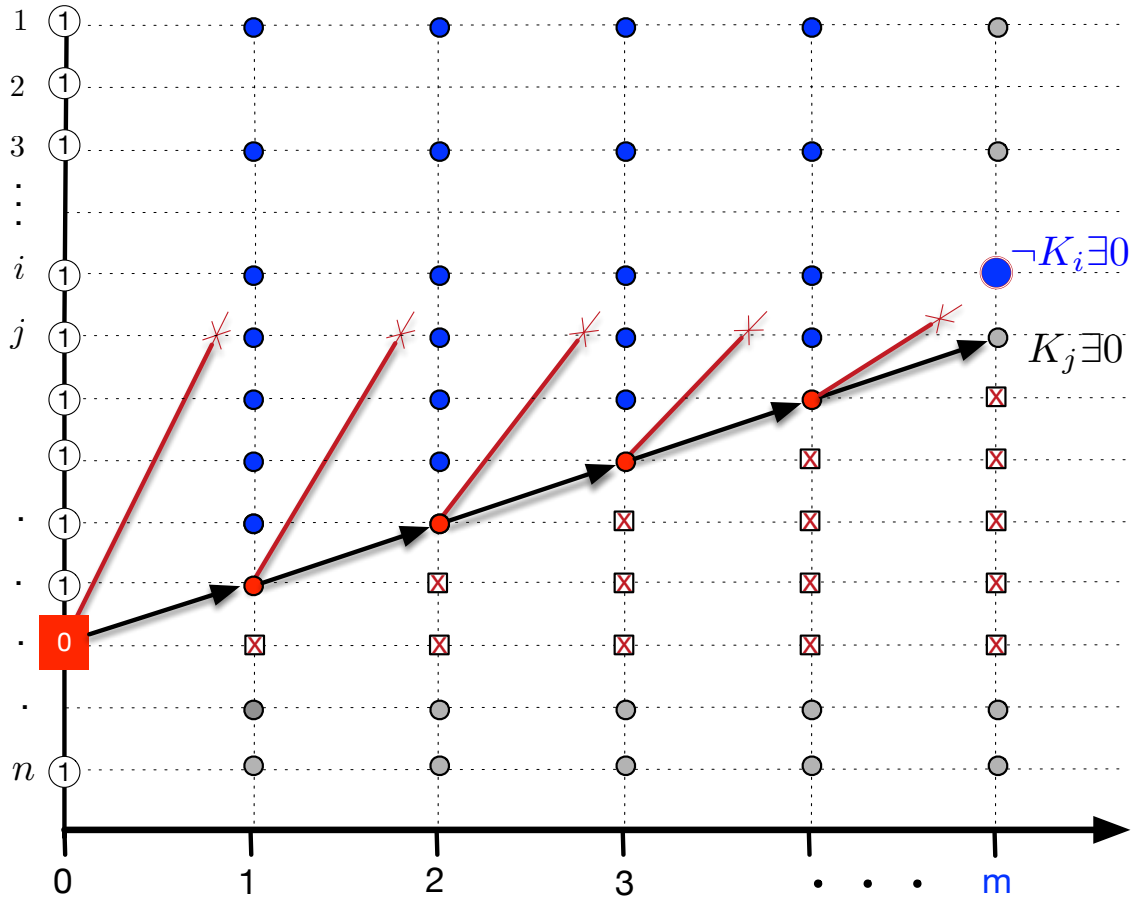
The Adventure of the Blue Carbuncle, 1892

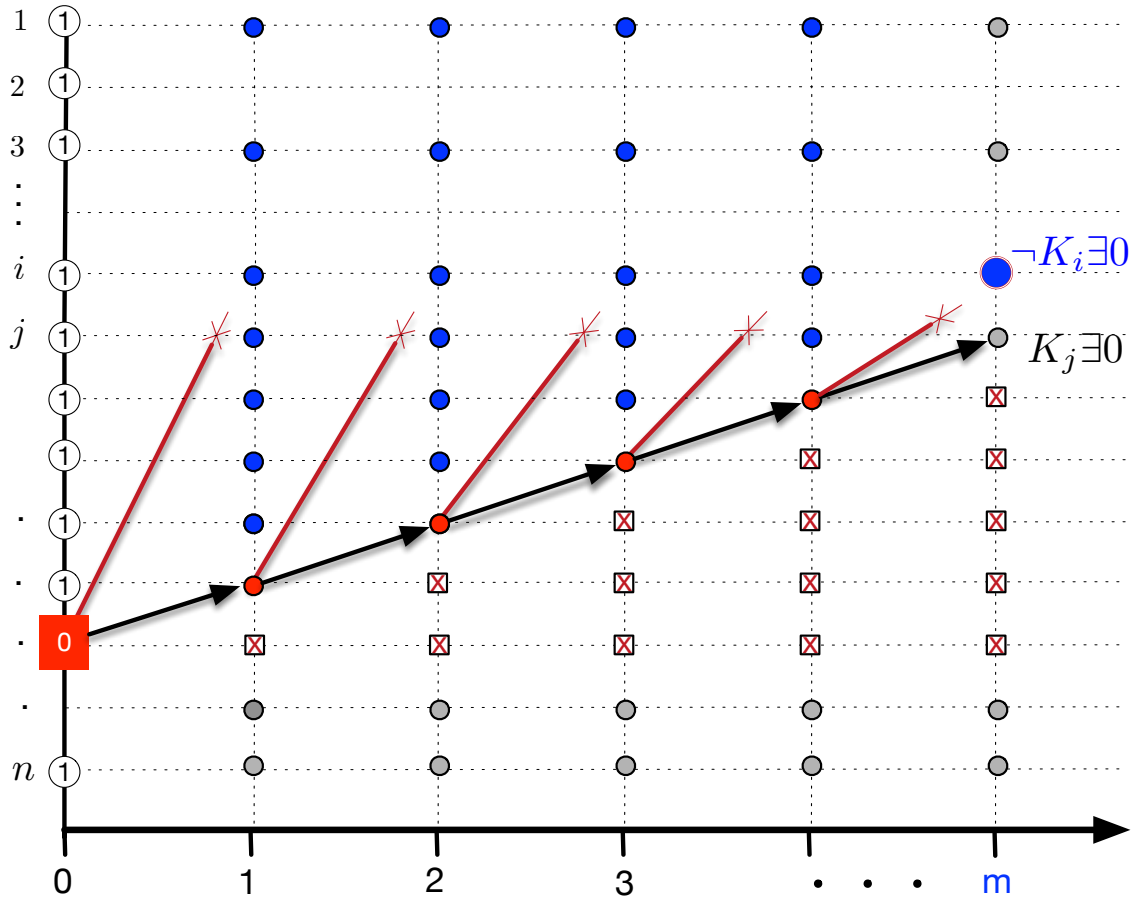
Implementing OPT_0

Protocol OPT_0 (for undecided process i):

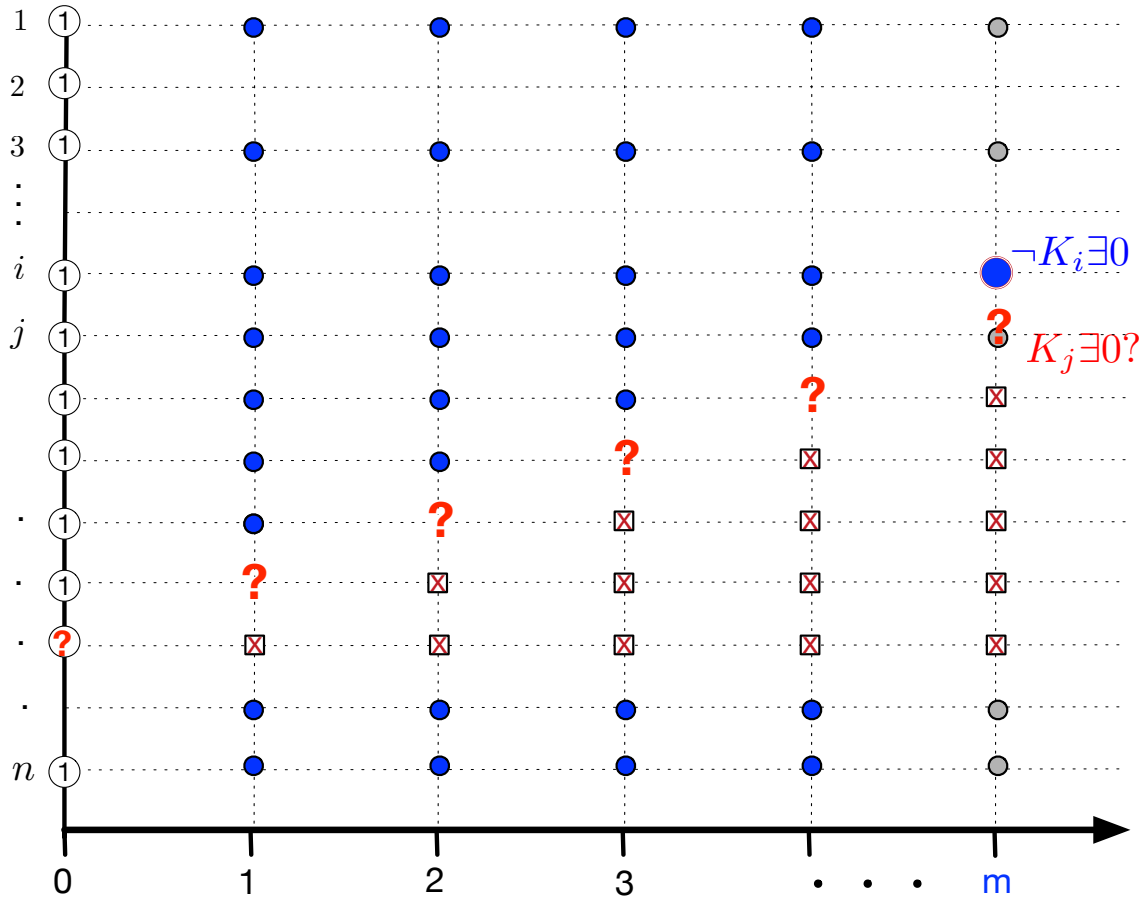
if	$K_i \exists 0$	**then** $\texttt{decide}_i(0)$
elseif	$K_i\big(\texttt{nobody\_knows} \exists 0\big)$	**then** $\texttt{decide}_i(1)$

To test for $K_i\big(\texttt{nobody\_knows} \exists 0\big)$, recall the analysis of $K_j \exists 0 \,\&\, \neg K_i \exists 0$

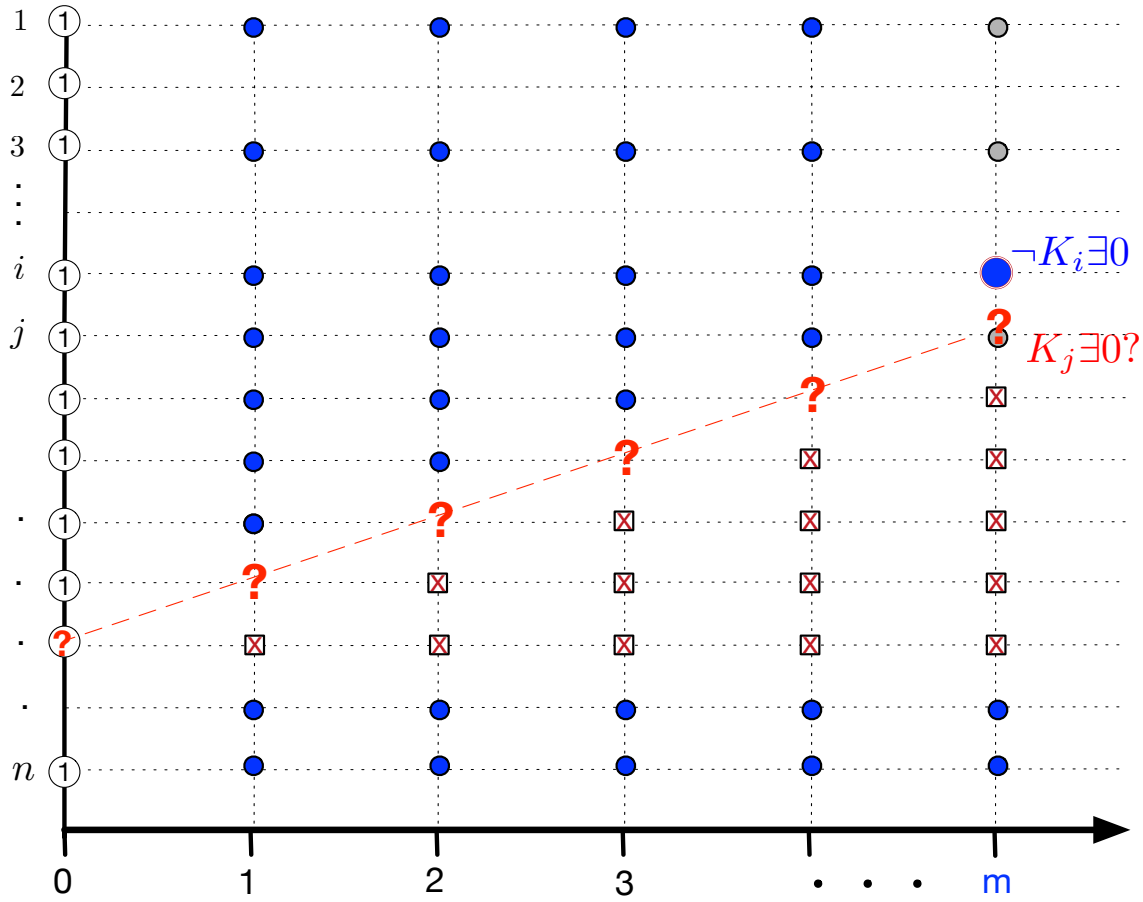

1
2
3
i
j
n
0
1
2
3
m
0
$\neg K_i \exists 0$
$K_j \exists 0$



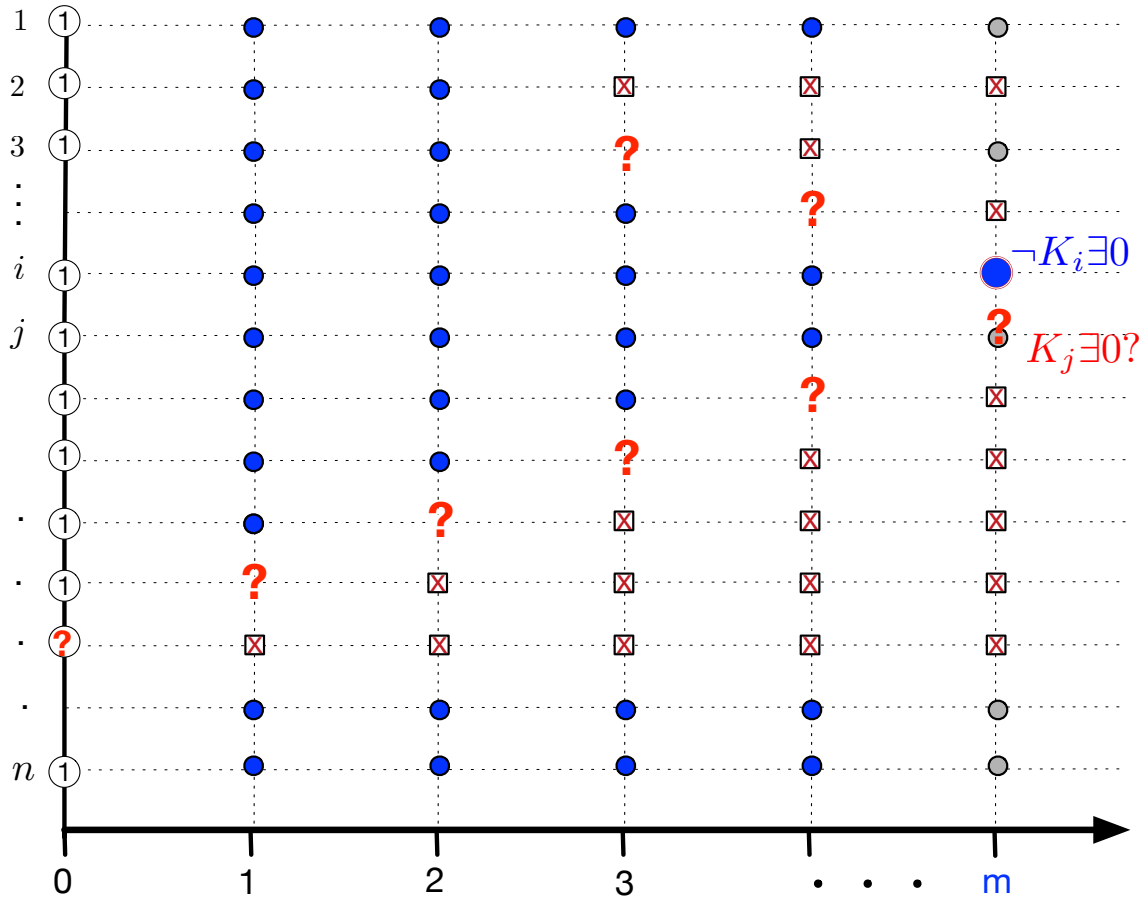
$K_i(\texttt{nobody\_knows}\exists 0)$ does not hold



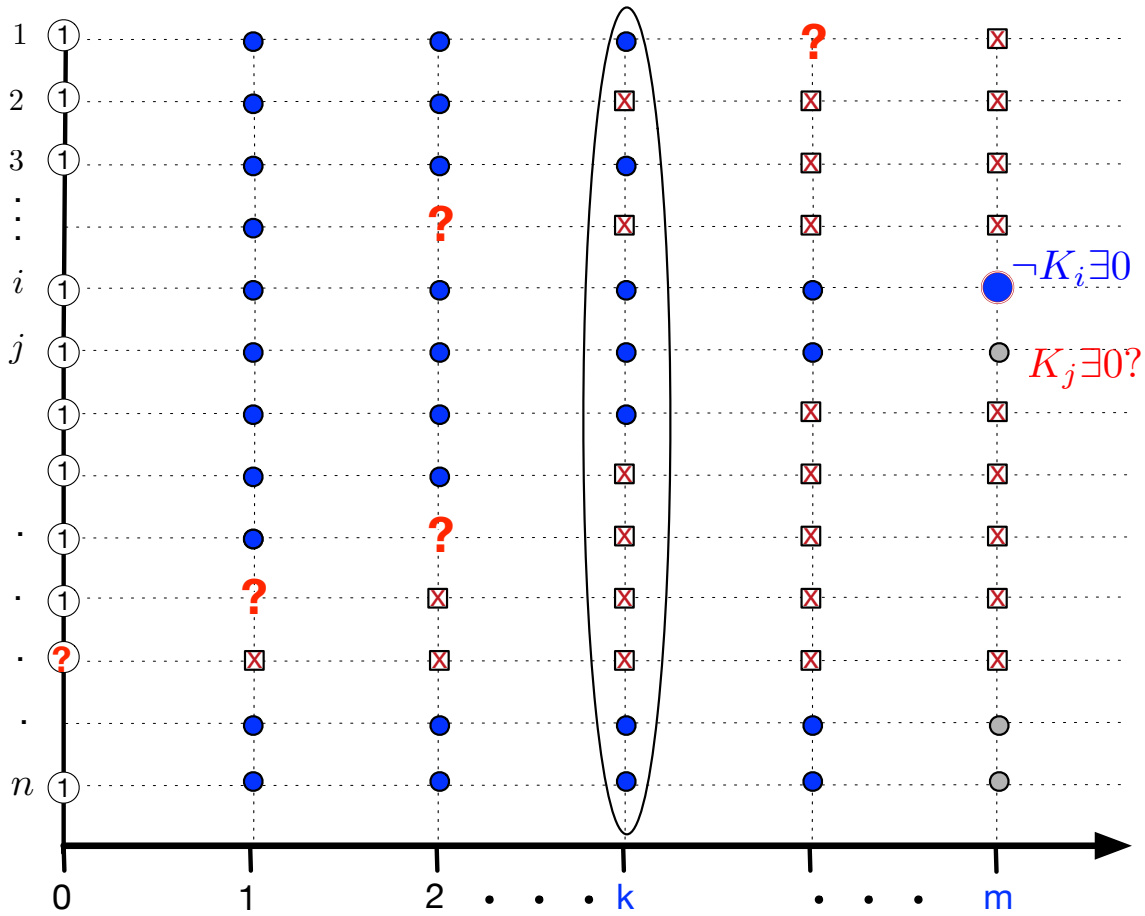
The world according to *i*



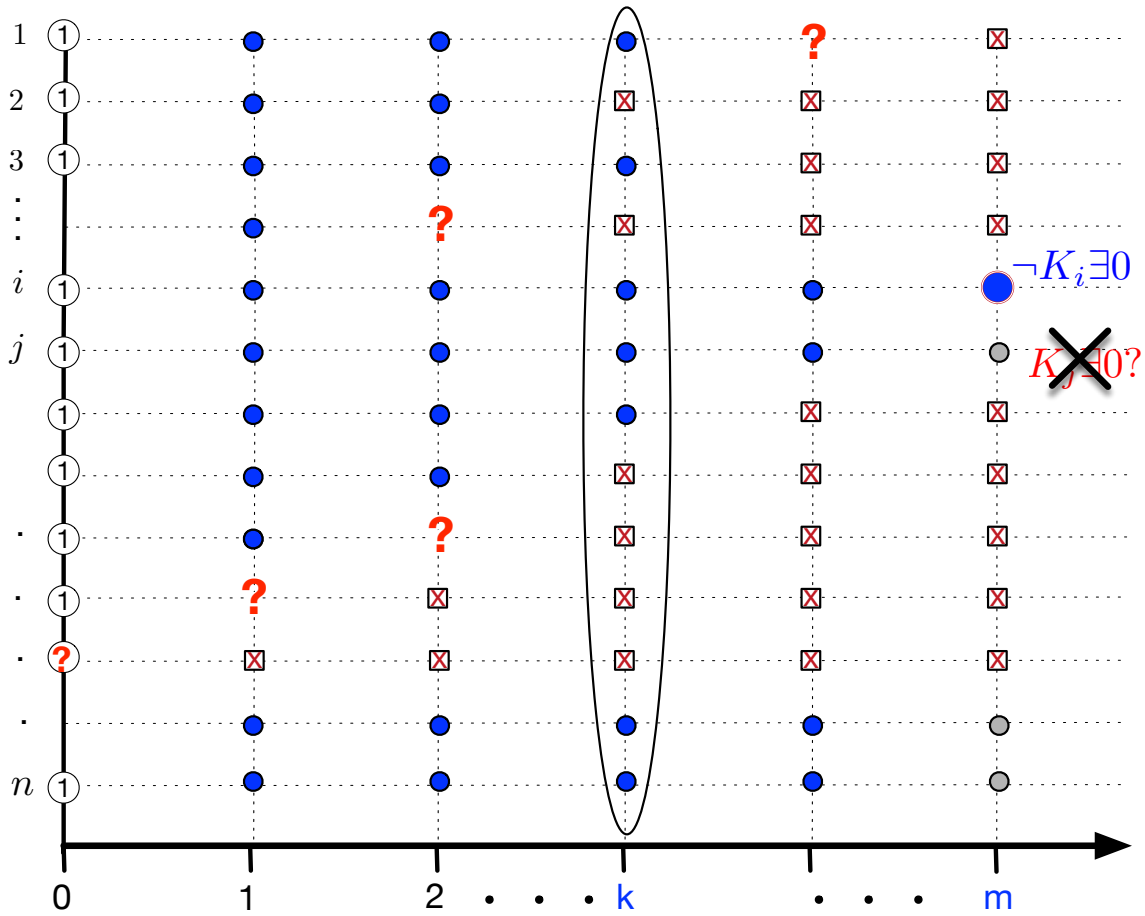
Process i's view contains a hidden path



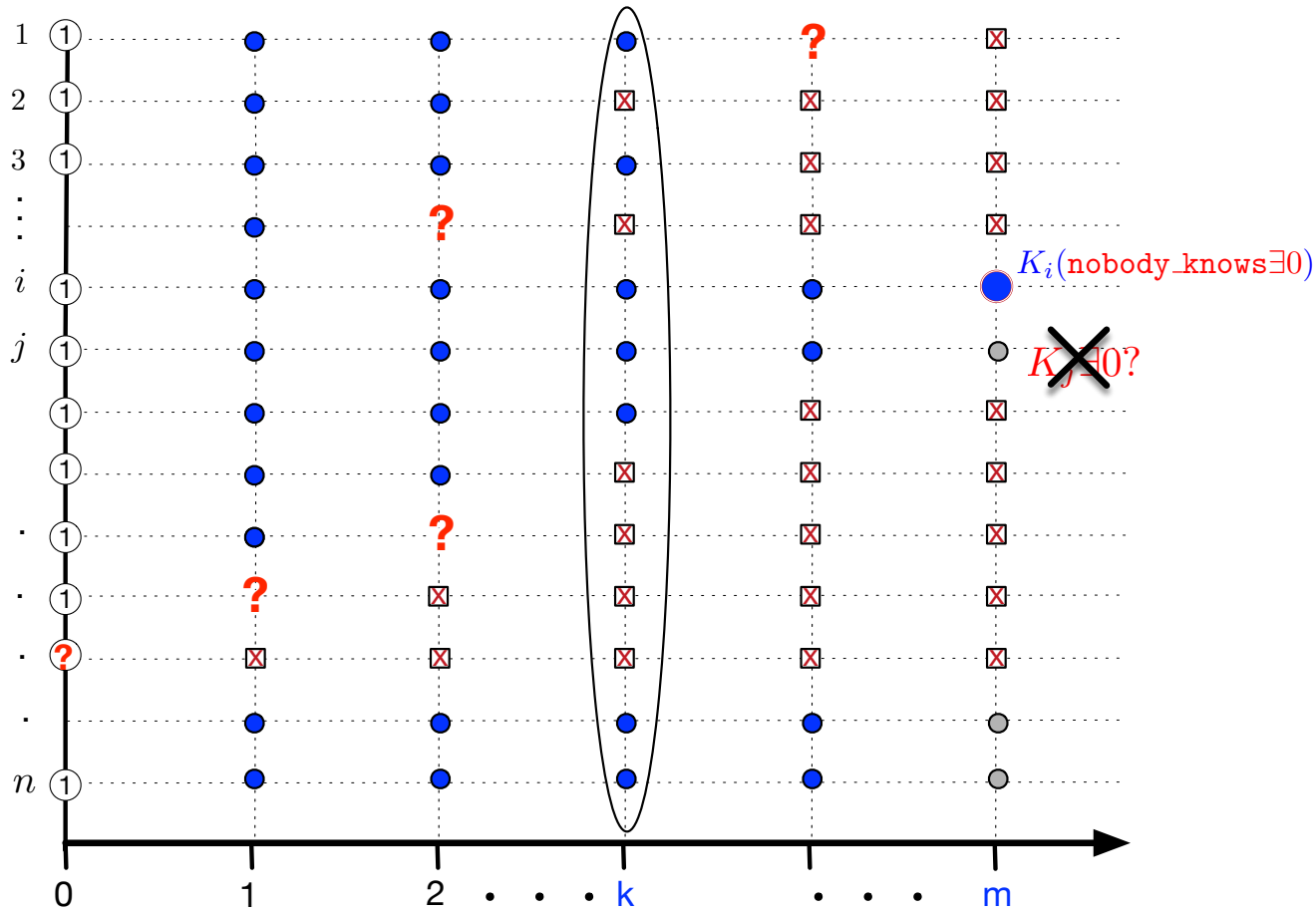
Each node $\langle h, k \rangle$ is **seen**, **crashed** or **hidden** w.r.t. $\langle i, m \rangle$



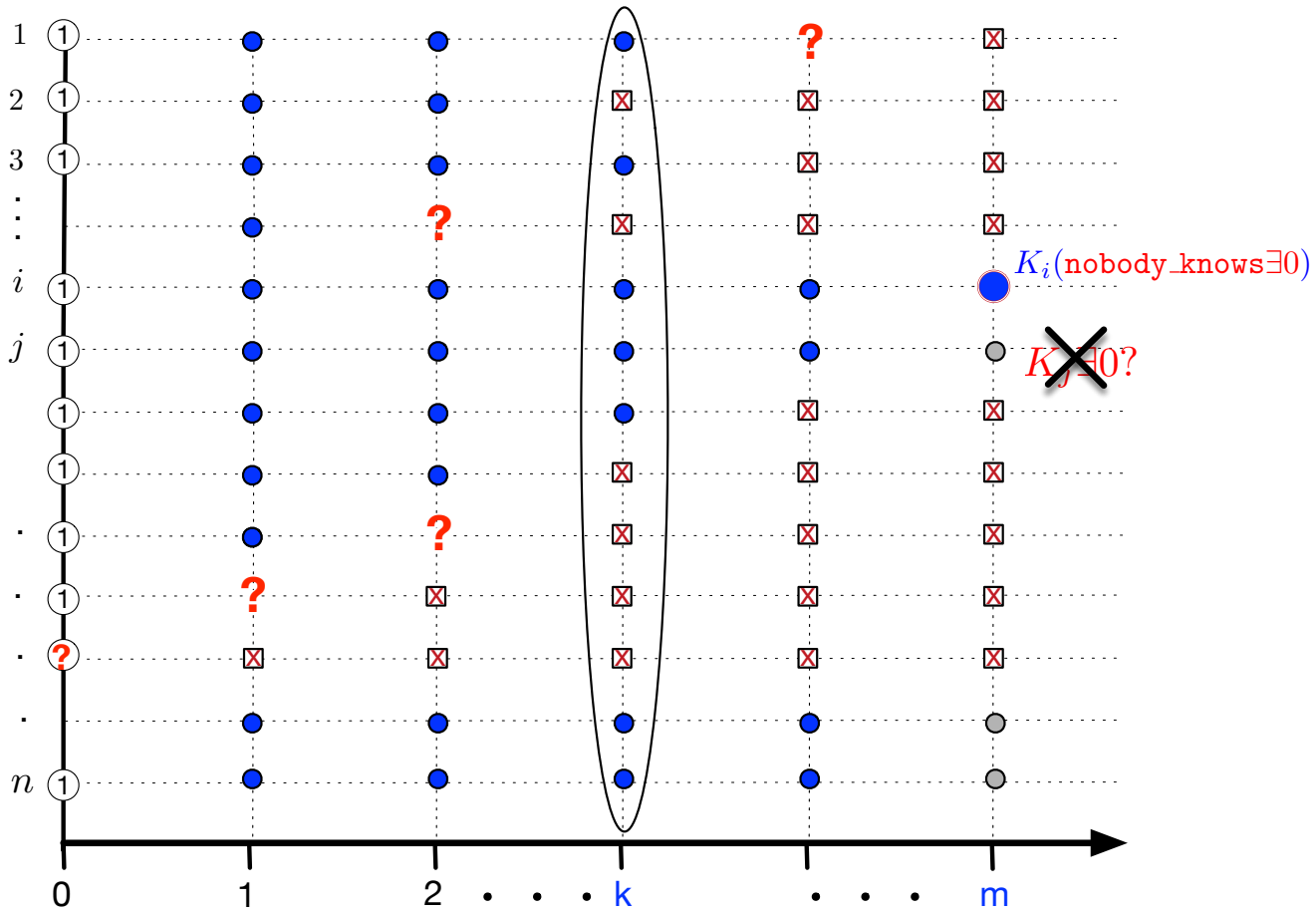
Time k is revealed at $\langle i, m \rangle$ if all nodes $\langle h, k \rangle$ are not hidden



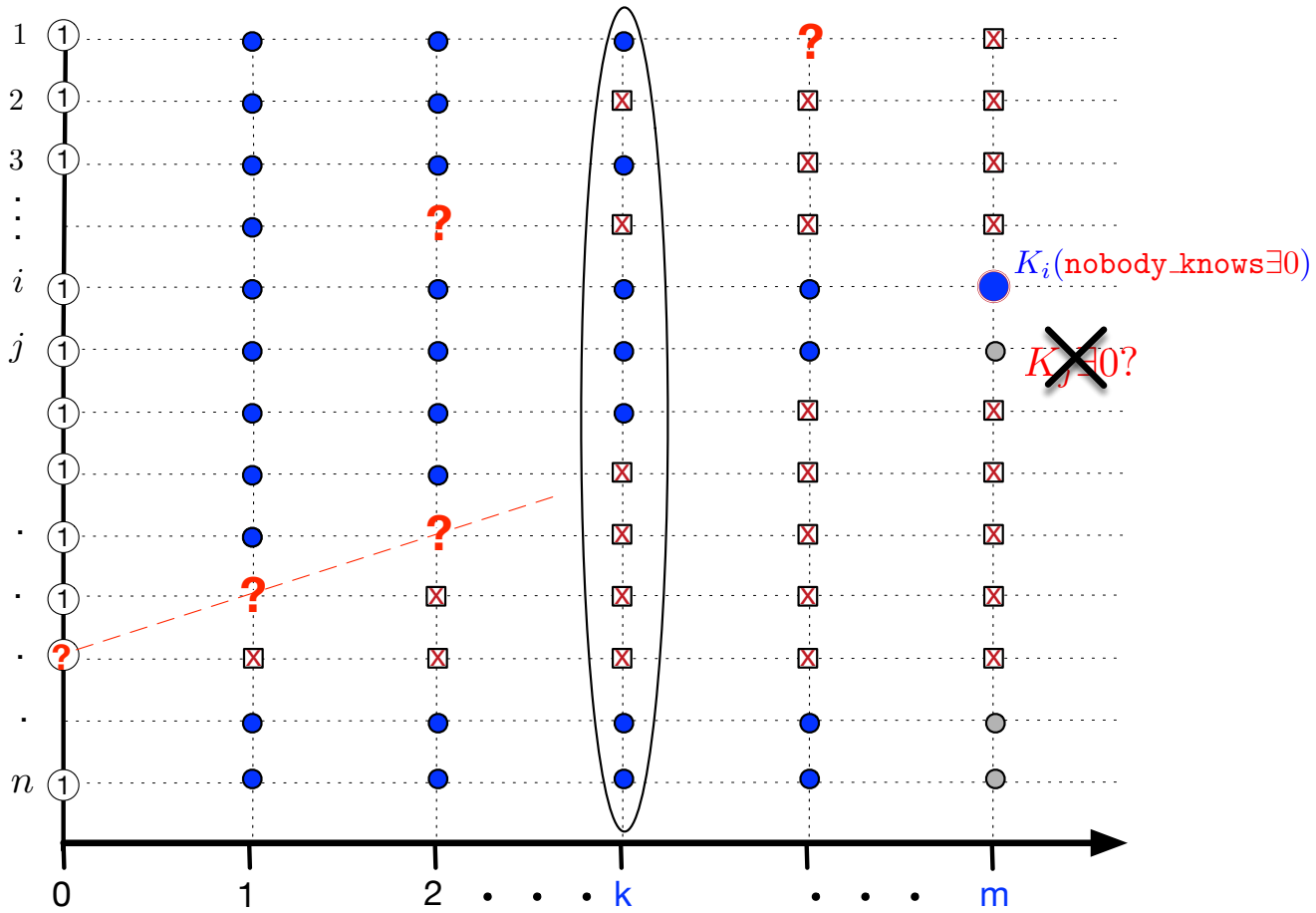
If time k is revealed and $\neg K_i \exists 0$ then $\neg K_j \exists 0$



If time k is revealed and $\neg K_i \exists 0$ then $K_i\big(\texttt{nobody\_knows}\exists 0\big)$



$K_i\big(\texttt{nobody\_knows}\exists 0\big)$ **iff** some time k is revealed and $\neg K_i \exists 0$



$K_i\big(\texttt{nobody\_knows}\exists 0\big)$ **iff** some time k is revealed and $\neg K_i \exists 0$

A Standard Protocol for OPT_0 [Castañeda, Gonczarowski & M. '14]

Standard OPT_0 (for undecided process i):

if	seen 0	**then** $\texttt{decide}_i(0)$
elseif	some time k is revealed to i	**then** $\texttt{decide}_i(1)$

Theorem (CGM)

- *OPT_0 dominates Q_0*
- *No consensus protocol dominates OPT_0 (it is unbeatable)*
- *OPT_0 implementable using $O(\log n)$ bit messages on average*

Another Example

Another Example: Majority Consensus

In addition to Decision, Validity and Agreement, we require:

Majority Rule: A correct process will

decide 0 if it discovers $\geq n/2$ of the votes are 0

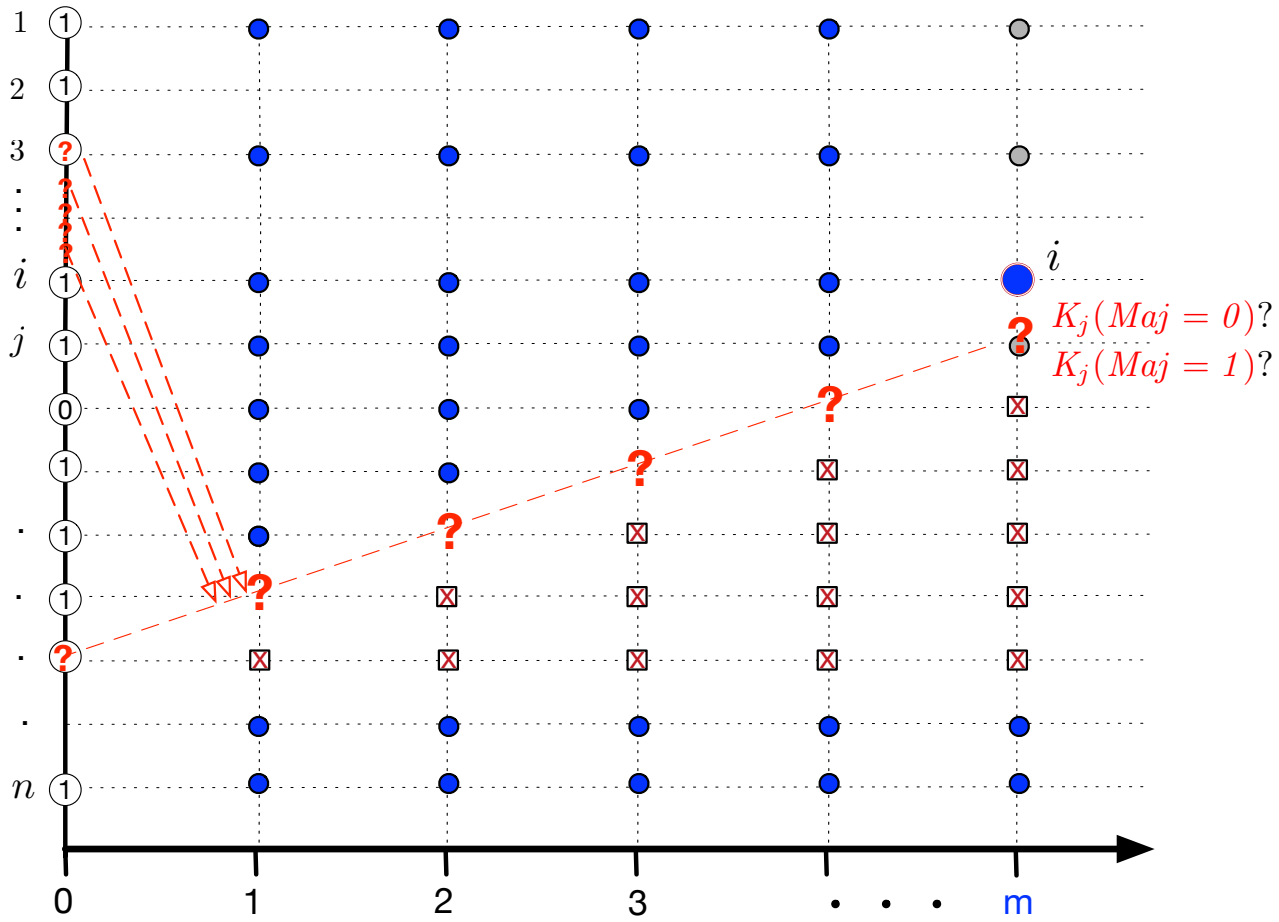
decide 1 if it discovers $> n/2$ of the votes are 1

All-case Optimal Majority Consensus

Standard OPT_{mc} (for undecided process i):

if	seen $\geq n/2$ votes of 0	**then** $\texttt{decide}_i(0)$
elseif	seen $> n/2$ votes of 1	**then** $\texttt{decide}_i(1)$
elseif	some time k is revealed to i & seen more votes for v than $1-v$	**then** $\texttt{decide}_i(v)$

Majority Consensus



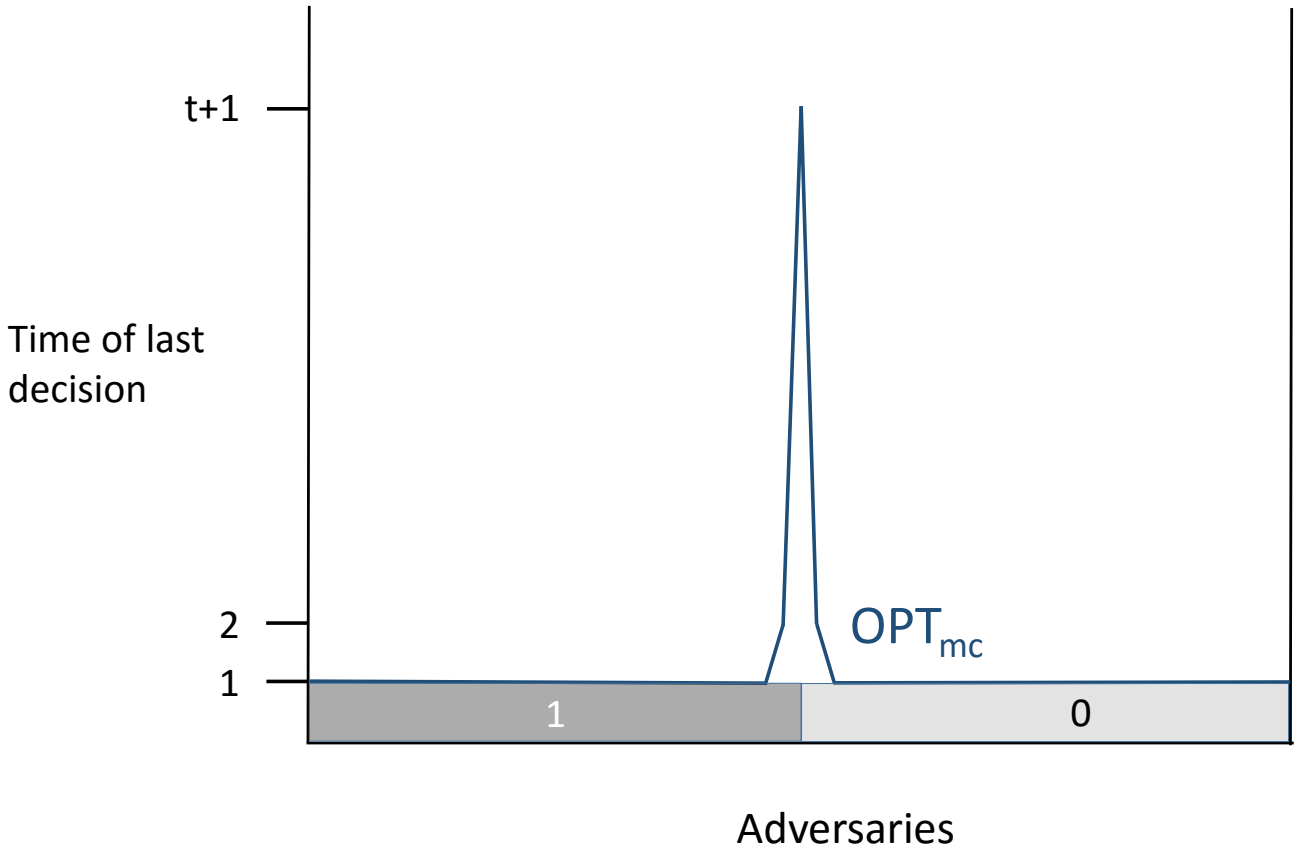
A hidden path can report all of the unknown votes

All-case Optimality

Theorem (CGM)

The protocol OPT_{mc}:

- *solves Majority Consensus*
- *dominates all protocols for Majority Consensus*
- *decides in $\leq f + 1$ rounds (aka "early stopping")*
- *is implementable using $O(\log n)$ bit messages on average*
- *Treats "0" and "1" fairly*


t+1
Time of last decision
2
1
OPT_{mc}
1
0
Adversaries

KoP Revisited

The **KoP** is a universal theorem for distributed systems

KoP applies more generally:

- Suppose that a legal system satisfies that Judge punishes X only if X committed the crime. By **KoP**, when deciding to punish the judge must know that X committed the crime.
- A jellyfish does not sting its own body. By **KoP** the jellyfish cell must know not my body when it launches a sting.
- Consider a (Turing) machine that will perform accept on a word $x \in \{0,1\}^*$ only if $x \in L$ for a given language L. Then the TM head must know, based on the current state and the letter seen on the tape, that $x \in L$.

KoP Revisited

The **KoP** is a universal theorem for distributed systems

KoP applies more generally:

- Suppose that a legal system satisfies that Judge punishes X only if X committed the crime. By **KoP**, when deciding to punish the judge must know that X committed the crime.
- A jellyfish does not sting its own body. By **KoP** the jellyfish cell must know not my body when it launches a sting.
- Consider a (Turing) machine that will perform accept on a word $x \in \{0,1\}^*$ only if $x \in L$ for a given language L. Then the TM head must know, based on the current state and the letter seen on the tape, that $x \in L$.

KoP Revisited

The **KoP** is a universal theorem for distributed systems

KoP applies more generally:

- Suppose that a legal system satisfies that Judge punishes X only if X committed the crime. By **KoP**, when deciding to punish the judge must know that X committed the crime.
- A jellyfish does not sting its own body. By **KoP** the jellyfish cell must know not my body when it launches a sting.
- Consider a (Turing) machine that will perform accept on a word $x \in \{0,1\}^*$ only if $x \in L$ for a given language L. Then the TM head must know, based on the current state and the letter seen on the tape, that $x \in L$.

Conclusions

KoP formally relates knowledge and action

Protocol specifications induce epistemic conditions

Knowledge is an essential aspect of distributed protocols

Knowledge-based analysis facilitates design of efficient protocols

Diverse applications including VLSI, Biology, real-time coordination and more

Thank You!