iCyPhy

Software Design for Cyber-Physical Systems

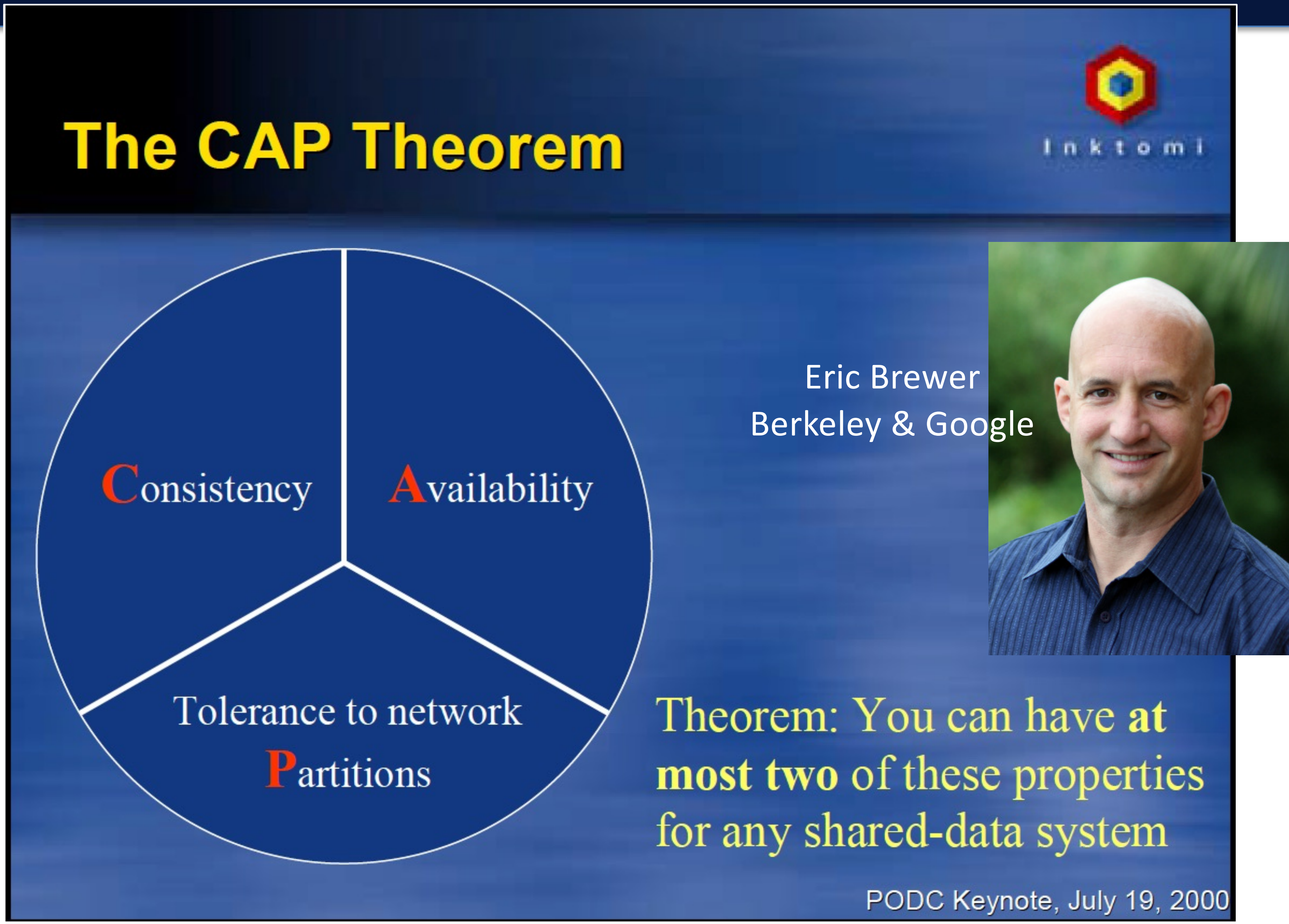
Edward A. Lee

Module 9: Consistency and Availability Tradeoffs

Technical University of Vienna
Vienna, Austria, May 2022

University of California, Berkeley

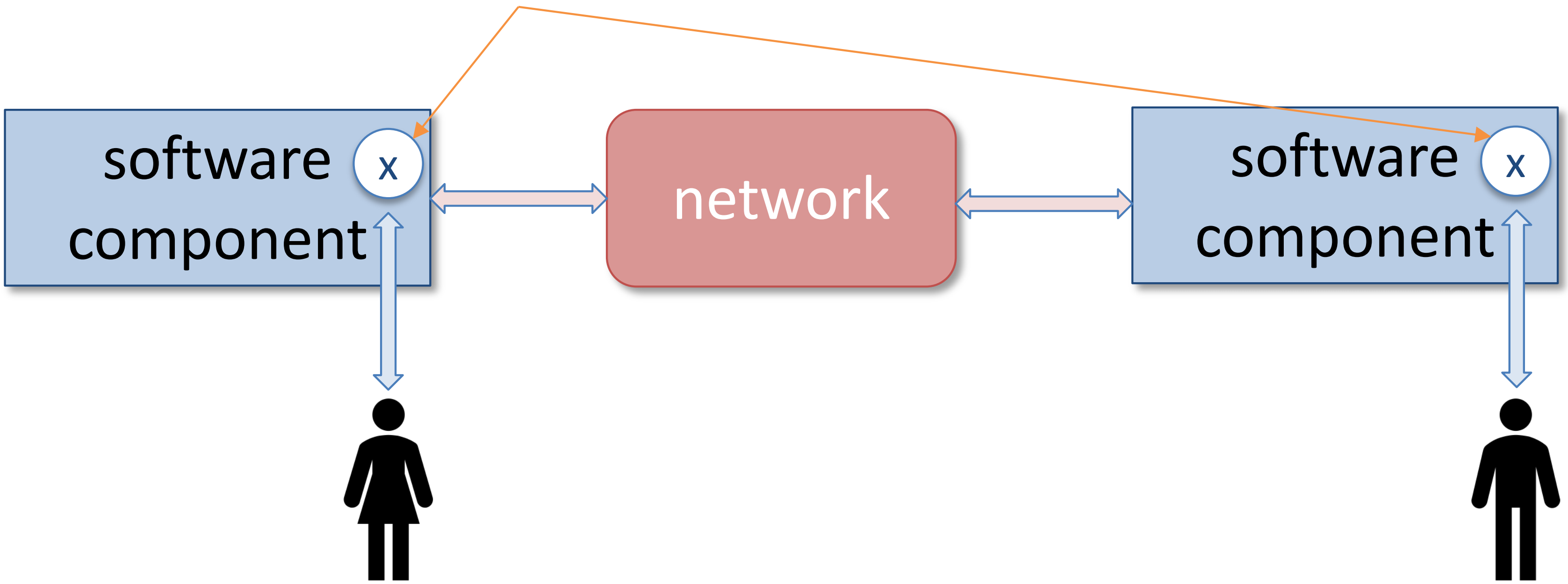
The CAP Theorem



Eric Brewer
Berkeley & Google

Consistency and Availability in Distributed Software

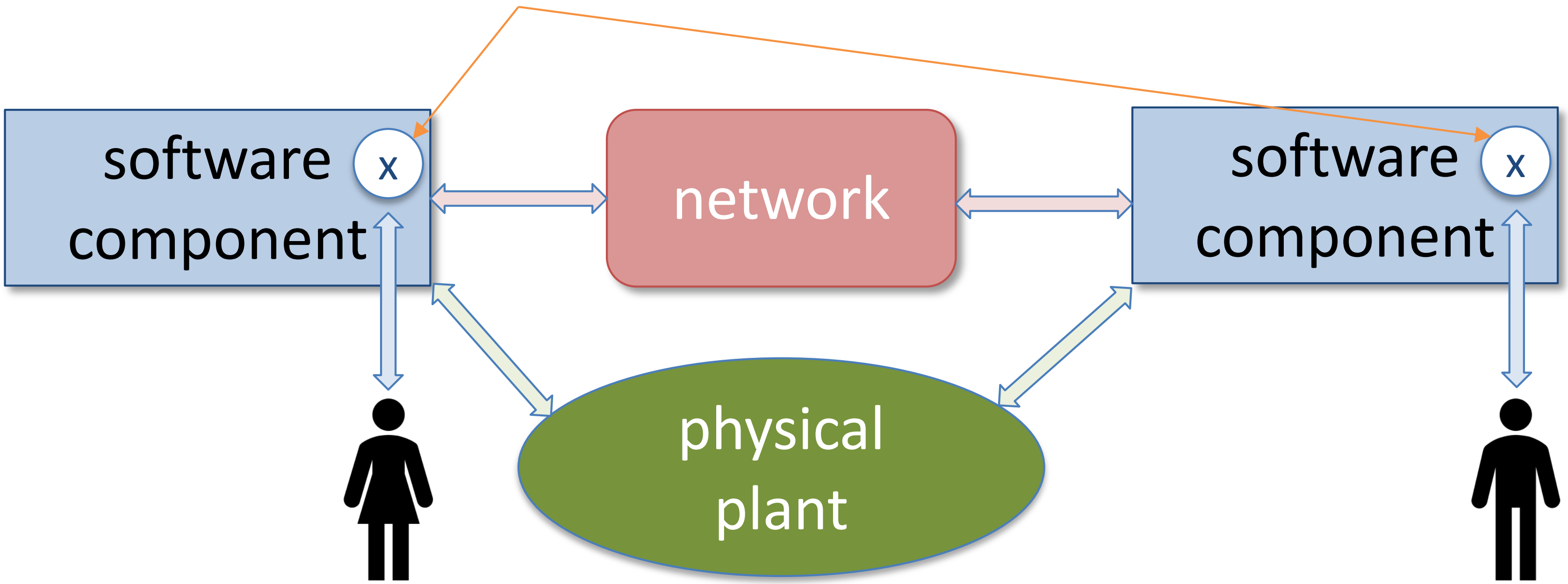
- **Consistency**: agreement on the values of shared variables



- **Availability**: ability to respond to reads and writes accessing those shared variables

Consistency and Availability in Cyber-Physical Systems

- **Consistency**: agreement on the values of shared variables and the state of the world



- **Availability**: ability to respond to reads and writes accessing those state variables

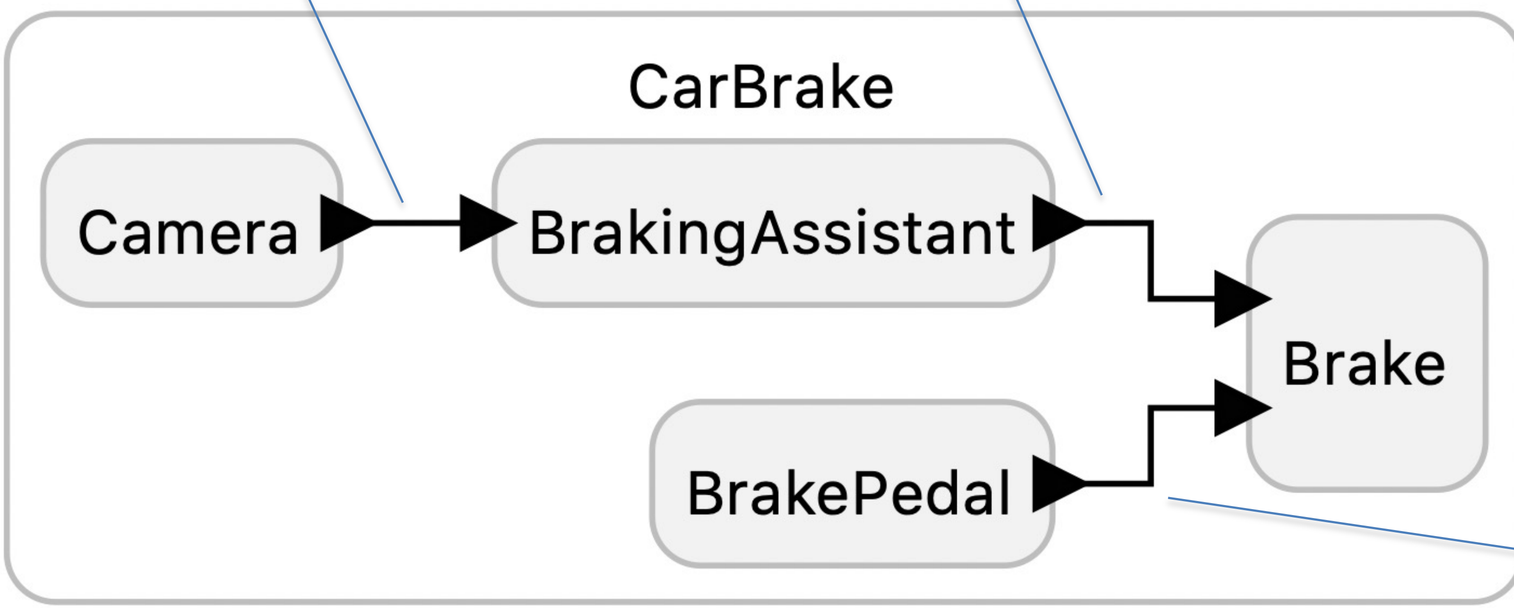
Availability or Consistency?

Denso autonomous braking demonstrating Advanced Driver-Assistance System (ADAS) in Oct. 2018 [Reported in The Daily Times]

Snapshot at time *T*

State of the world at time *T*

A software architecture:



State of the world at time *T* + ε

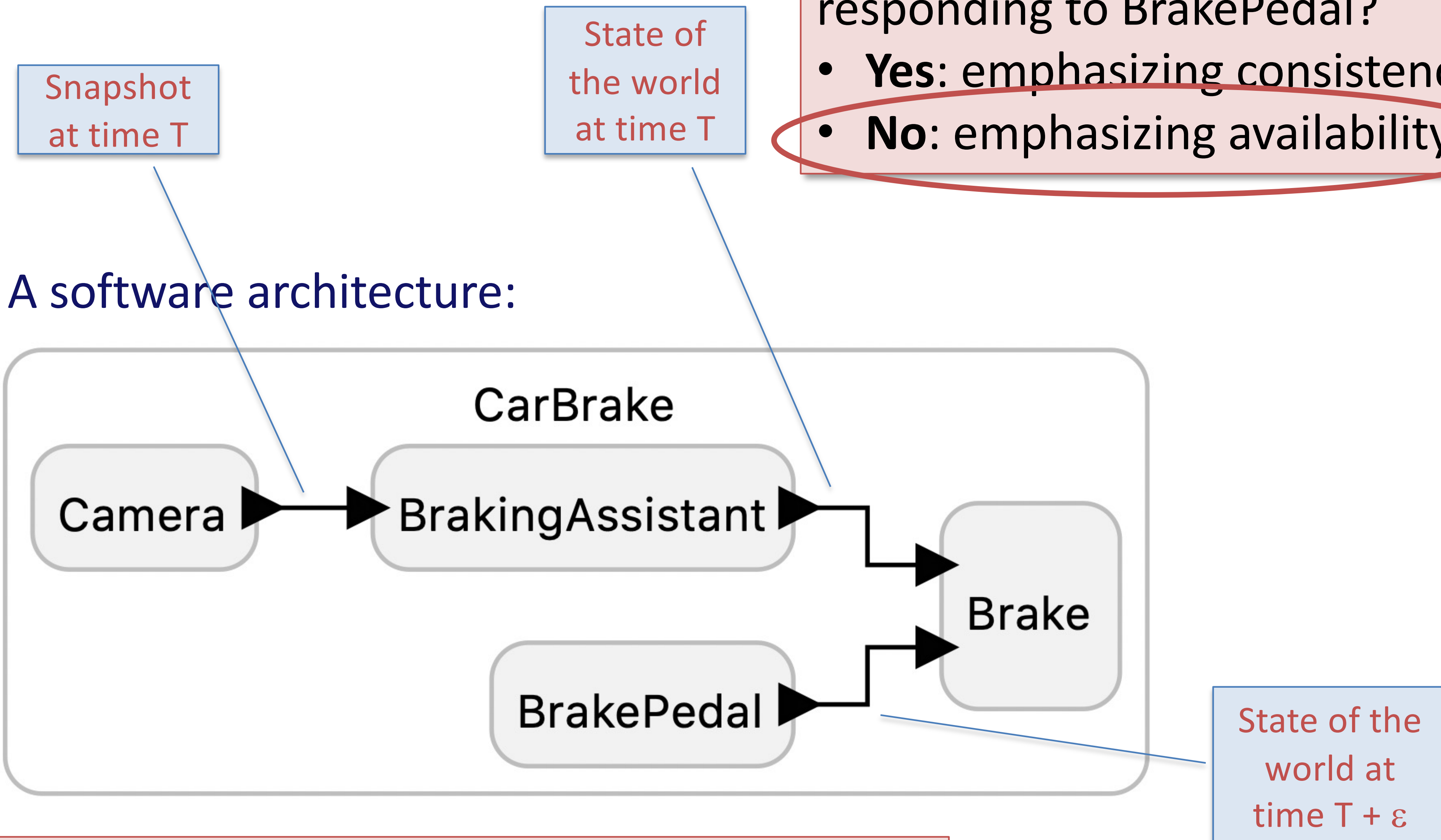
Thanks to Christian Menard (TU Dresden) for this example.

Availability or Consistency?

Should the Brake component wait for the analysis of the BrakingAssistant before responding to BrakePedal?

- **Yes**: emphasizing consistency
- **No**: emphasizing availability

A software architecture:



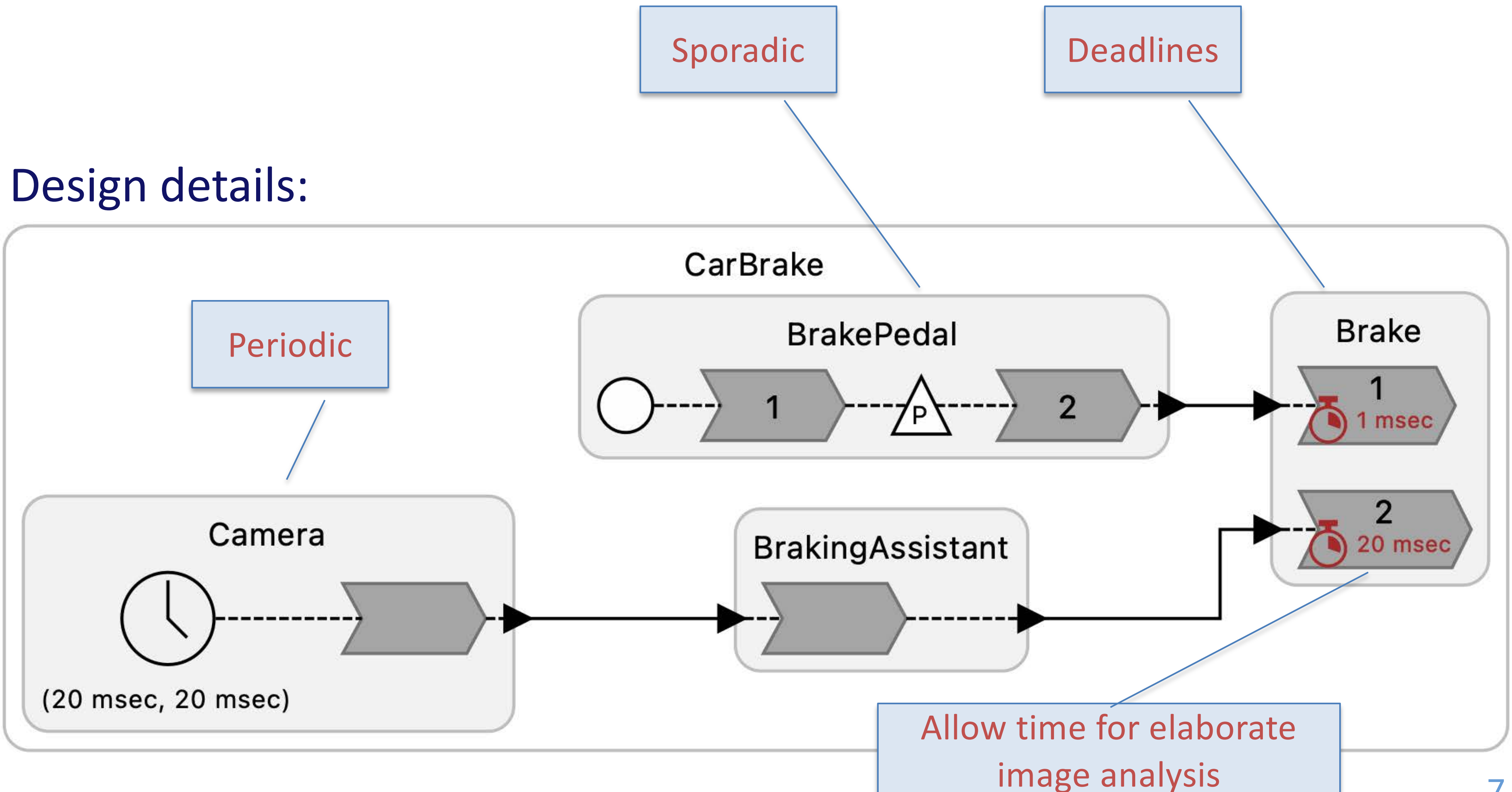
Thanks to **Christian Menard** (TU Dresden) for this example.

Availability or Consistency?

Discussion: Does this design emphasize availability over consistency?

When are deadline violations likely to occur?

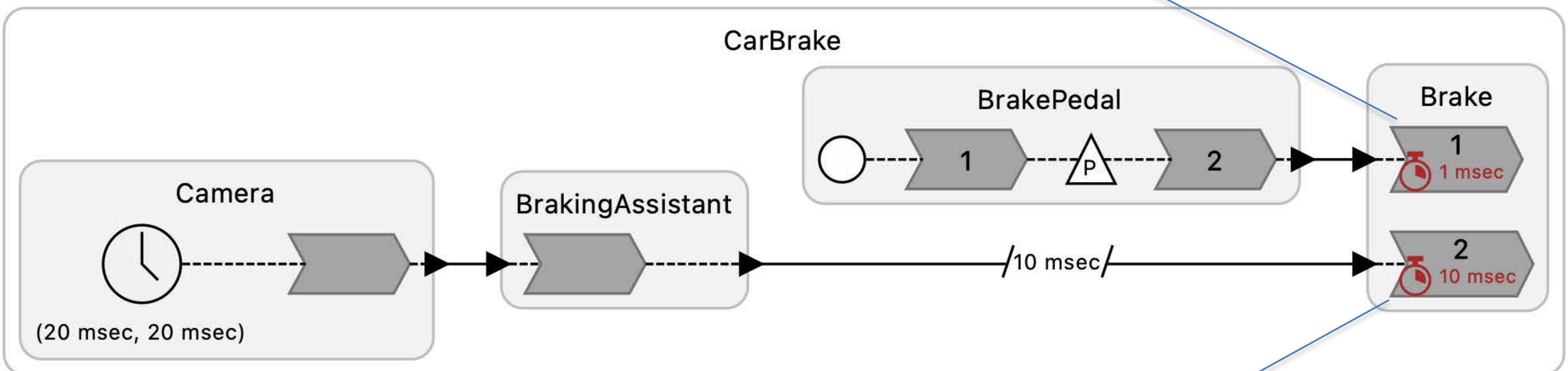
Design details:



An Alternative Design

Discussion: What should the deadline violation handlers do?

As long as the BrakingAssistant takes less than 10 msec, it will not block reactions to the BrakePedal.



Discussion: This design also puts a *minimum* delay on the response to the BrakingAssistant.
Is this this a good idea?
How could this minimum delay be avoided?

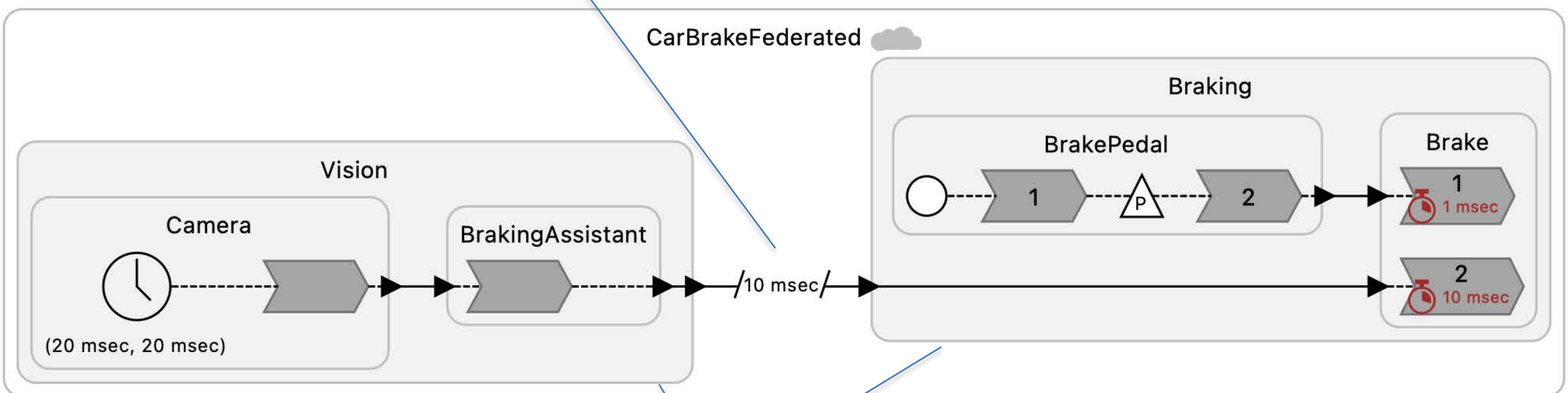
Response time is specified to be between 10 and 20 msec

Federated Design

Discussion: Should this use centralized or decentralized coordination?

Discussion: Under decentralized coordination, what should STP violation handlers do?

Explicit inconsistency



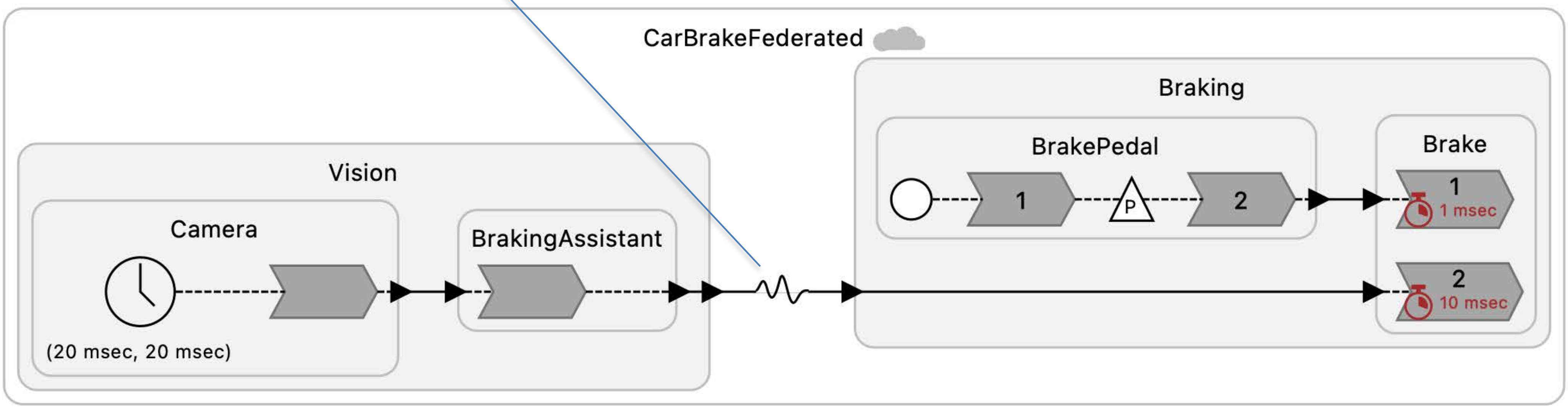
These now run on separate processes (and maybe separate processors)

Note that the deadline/STP violation handlers provide for **continuous testing**.

Asynchronous Design

Physical connection:
`vision.trigger_brake ~> braking.brake_assistant`

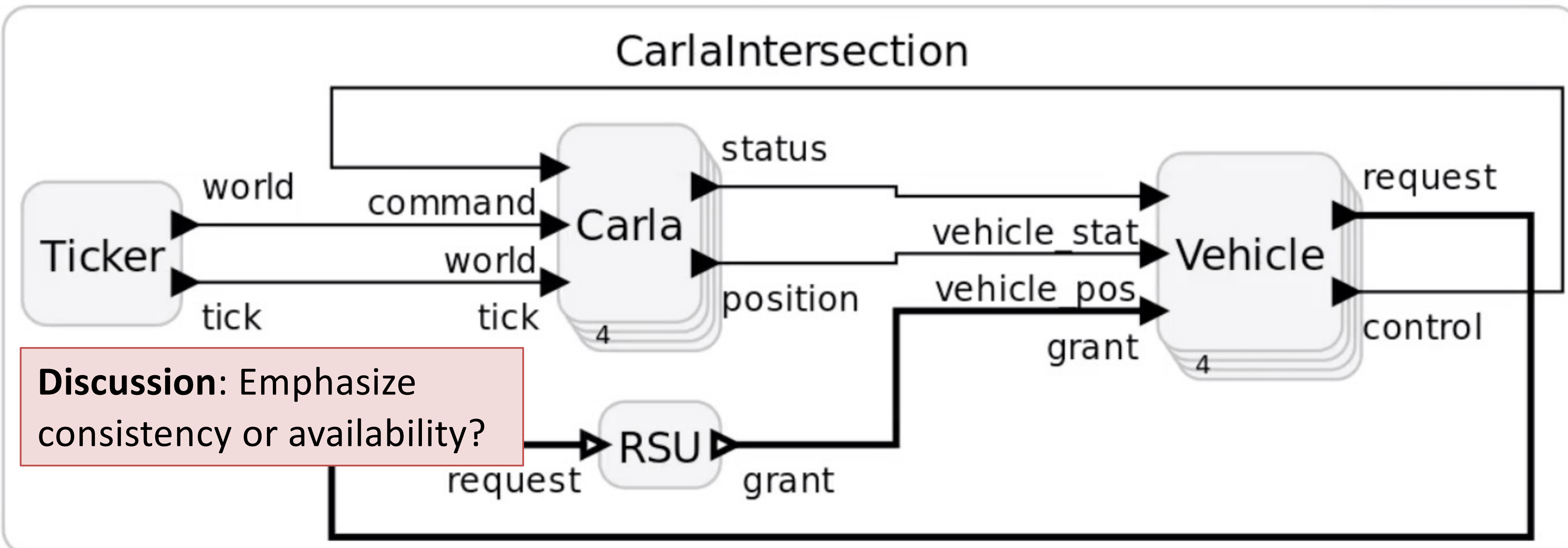
Discussion: Advantages and disadvantages?



Network failures and failures of the Vision subsystem are undetectable.

Intersection

- **Consistency**: agreement on the state of the intersection.
- **Availability**: ability to enter the intersection.

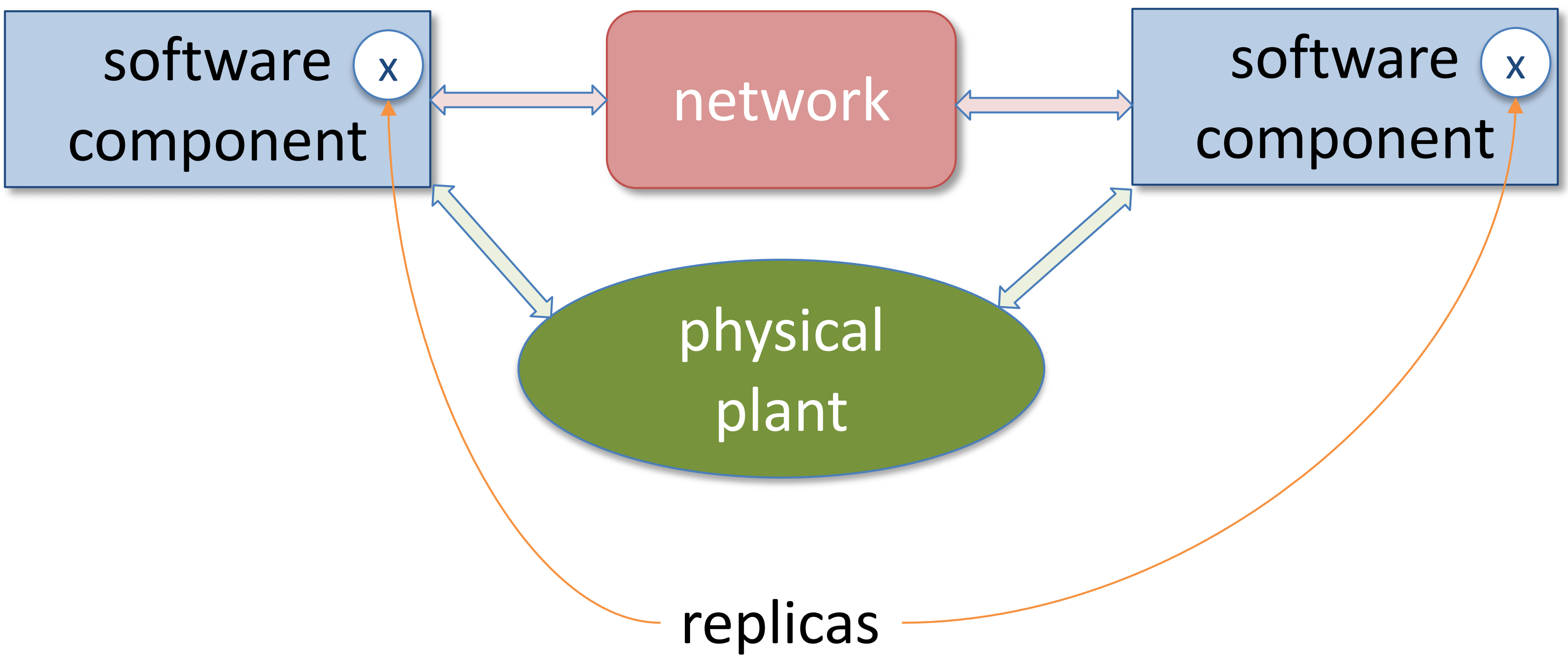


Discussion: Emphasize consistency or availability?

Thanks to **Soroush Bateni** (UT Dallas) and **Ravi Akella** (Denso) for this example.

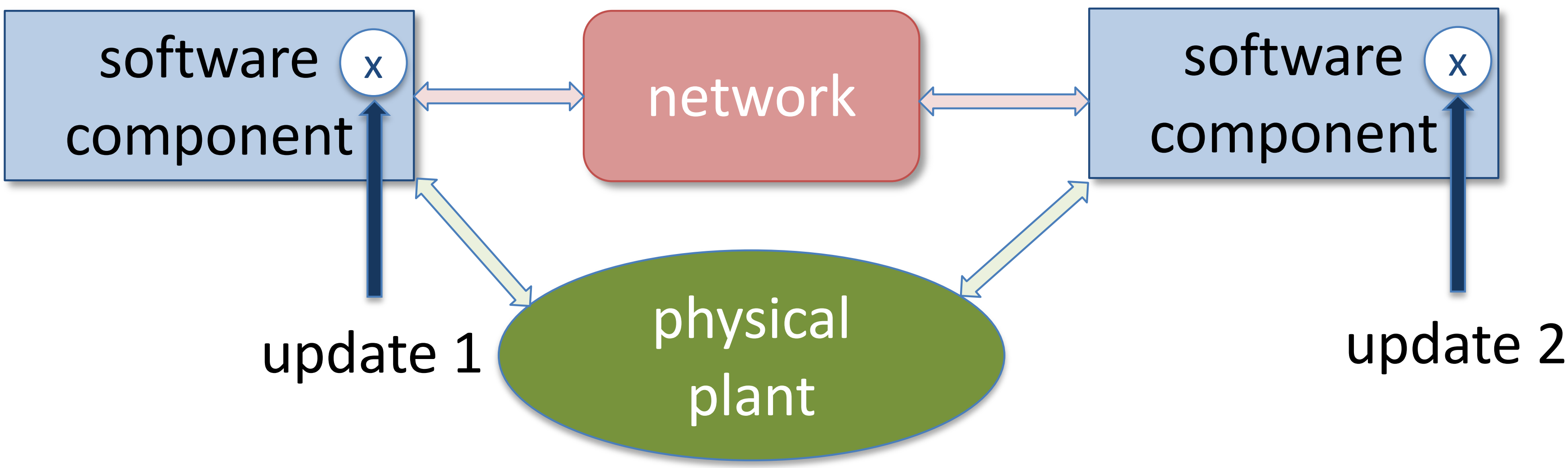
The Need For Replicas

If there are shared variables, then, to tolerate network latency, replication is necessary.



Ordering Updates

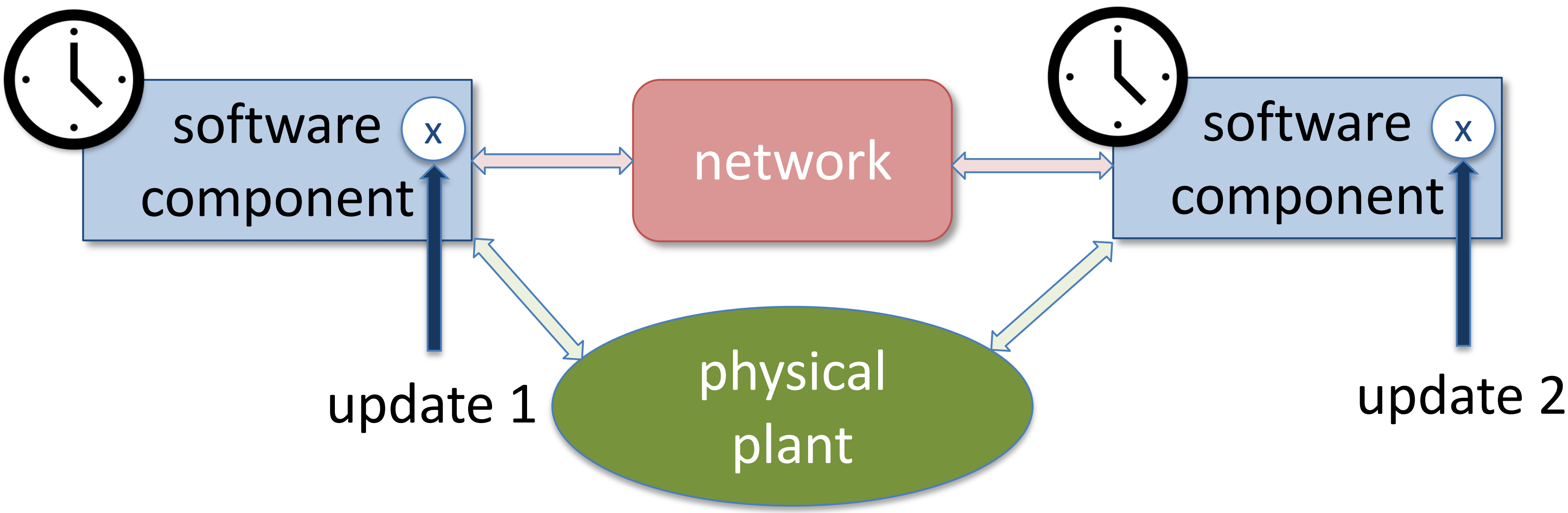
Assume updates can occur in multiple places:



Consistency requires agreement on the order of these updates.

Physical Time is Imperfect

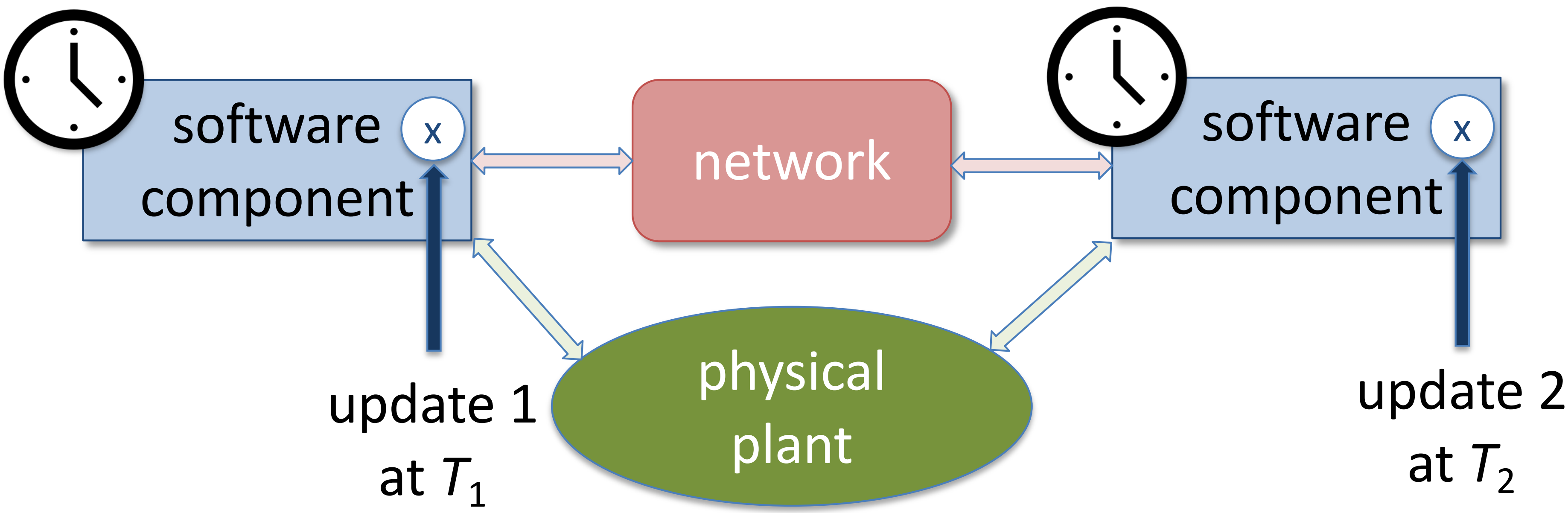
We have *imperfect* measurements of time.



With clock synchronization (albeit imperfect), physical time can used to assign a logical time.

Timestamps

We have *imperfect* measurements of time.



With clock synchronization (albeit imperfect), physical time can used to assign a logical time.

Timestamps in Use

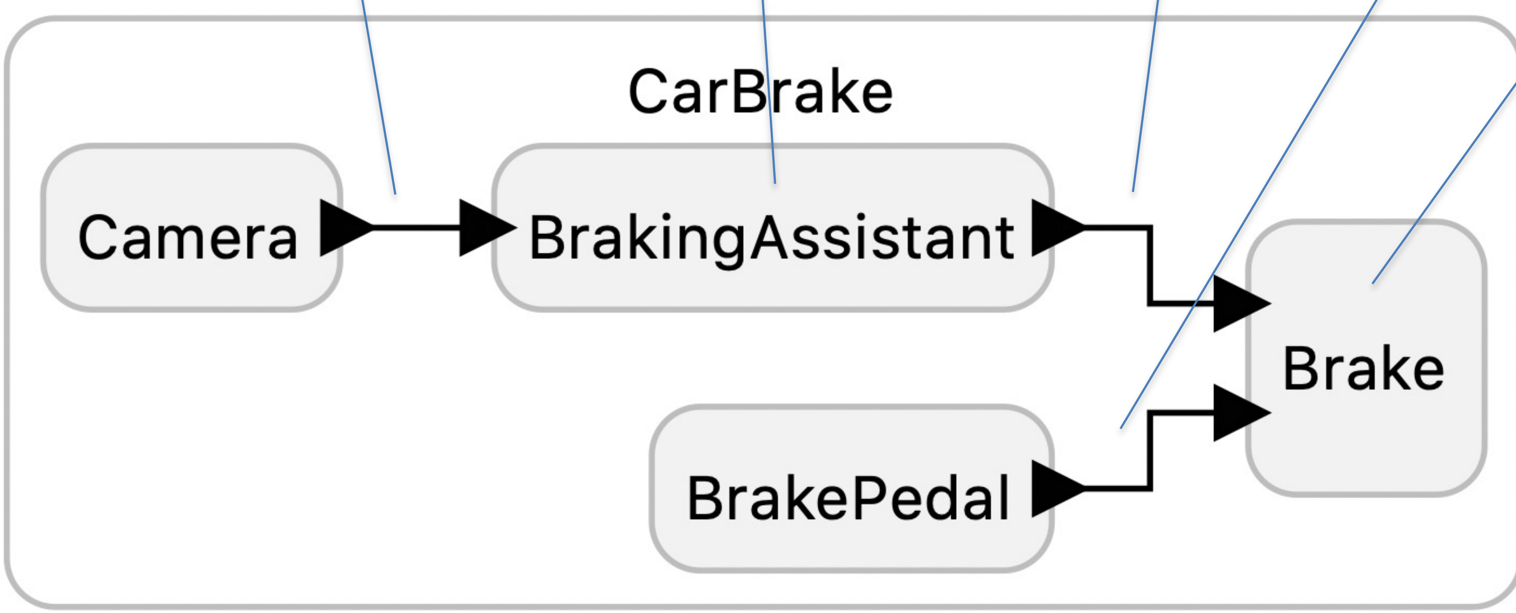
Snapshot assigned **logical time** $t_1 = T_1$, where T_1 is from the local clock

Logically instantaneous operation

Timestamp is still t_1

Event is assigned **logical time** $t_2 = T_2$, where T_2 is from the local clock

Consistency requires seeing events in timestamp order



Enforcing consistency here reduces availability!

Thanks to Christian Menard (TU Dresden) for this example.

Recent Result: CAL Theorem

Quantifying and Generalizing the CAP Theorem

Edward A. Lee[1[0000–0002–5663–0584]], Soroush Bateni[2[0000–0002–5448–3664]], Shaokai Lin[1[0000–0001–6885–5572]], Marten Lohstroh[1[0000–0001–8833–4117]], and Christian Menard[3[0000–0002–7134–8384]]

arXiv:2109.07771v1 [cs.DC] 16 Sep 2021

Theorem 1. *Given a trace as defined in Section 3.2, the unavailability in Definition 2 at process i is, in the worst case,*

$$\bar{A}_i = \max\left(O_i, \max_{j \in N}(\mathcal{L}_{ij} - \bar{C}_{ij})\right), \tag{17}$$

where O_i is the processing offset given by Definition 3, $\mathcal{L}_{ij}$ is apparent latency in Definition 4 (which includes O_j), and $\bar{C}_{ij}$ is the inconsistency of Definition 1 .

Causation

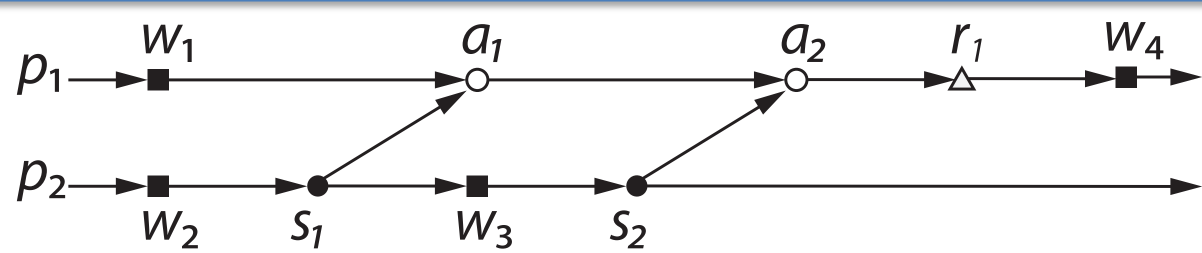
Event e_1 **causally effects** e_2 if e_2 cannot behave as if e_1 had not occurred.

Event e_1 **counterfactually causes** e_2 if e_2 will not occur if e_1 had not occurred.

In both case, we write $e_1 \rightarrow e_2$

See Lee (2020), *The Coevolution*, Chapter 11, for subtleties around causation.

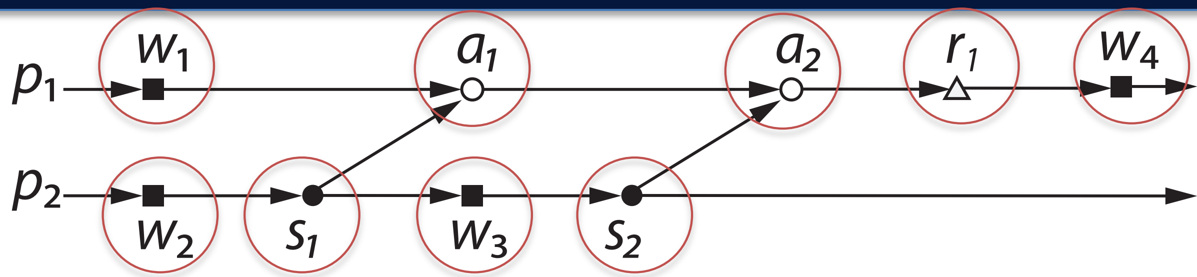
A Process Model



Event types:

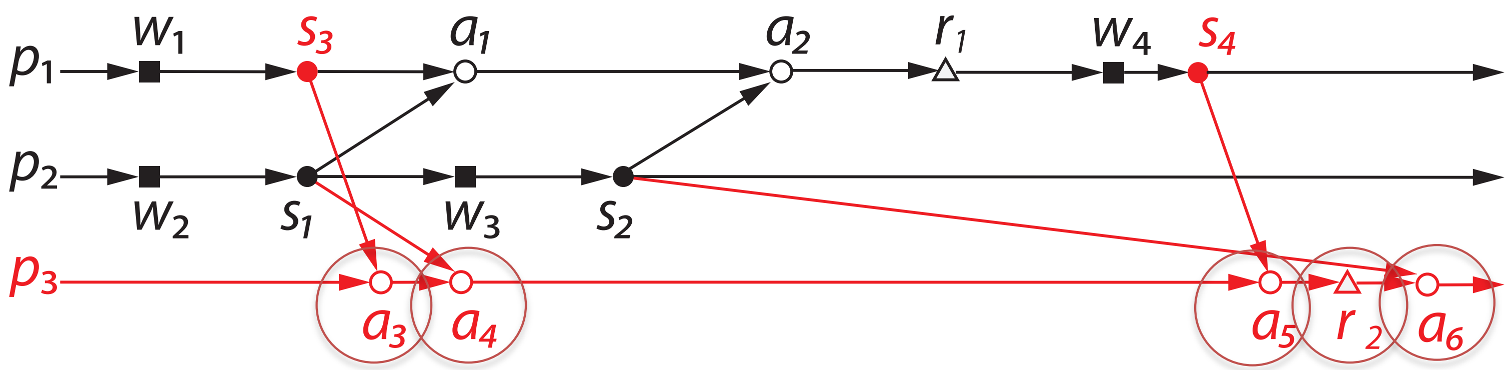
- w_x : Merge a value with the local replica of x.
- r_x : Read the value of the local replica of x.
- s_x : Send the value of the local replica of x to some set of other processes.
- a_x : Accept a new value for x and merge it with the local replica.

A Bulletin Board



- w_1: Joe posts a picture of Sally at a recent party by writing to a local copy.
- w_2: Sally posts that her son Billy is missing, writing to a local copy.
- s_1: Sally's message is sent to Joe's process.
- a_1: Joe's machine receives the message and updates his local copy.
- w_3: Sally posts that her son has been found (on the local copy).
- s_2: Sally's message is sent to Joe's process.
- a_2: Joe's machine receives the second message and updates his local copy.
- r_1: Joe reads Sally's messages.
- w_4: Joe posts "That's good news, a relief."

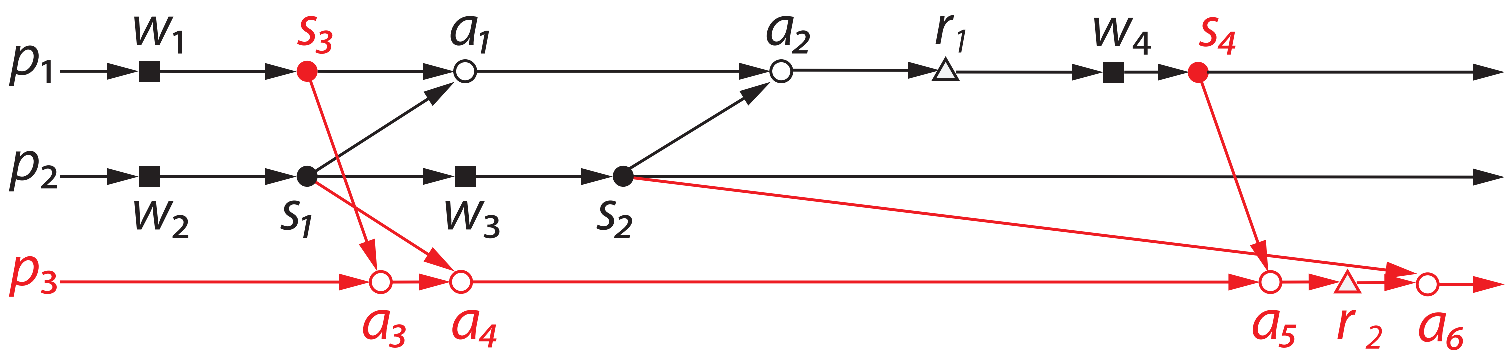
Bulletin Board Observer



- a_3: Akosh receives Joe's picture of Sally.
- a_4: Akosh receives Sally's post that her son Billy is missing.
- a_5: Akosh receives Joe's post "That's good news, a relief.".
- r_2: Akosh reads the posts so far.
- a_6: Akosh receives Sally's post that her son has been found.

This sequence violates **causal consistency.**

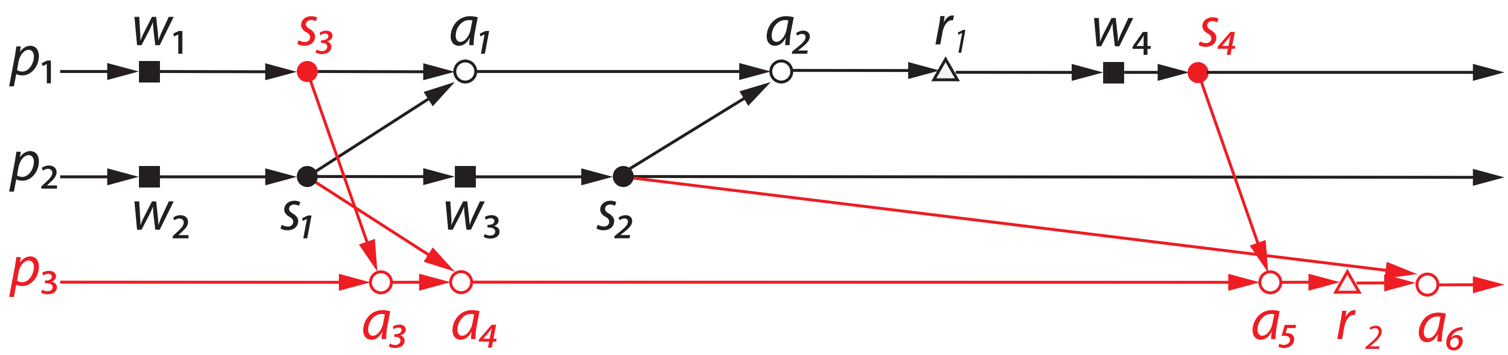
Causality Relation



Formally, the **causality relation** is the smallest *transitive* relation such that $e_1 \rightarrow e_2$ if e_1 precedes e_2 in a process, or e_1 is the sending of a value in one process (event type s_x) and e_2 is the acceptance of the value in another process (event type a_x).

Schwarz and Mattern (1994) "Detecting causal relationships in distributed computations: in search of the holy grail." *Distributed Computing*.

Causal Consistency



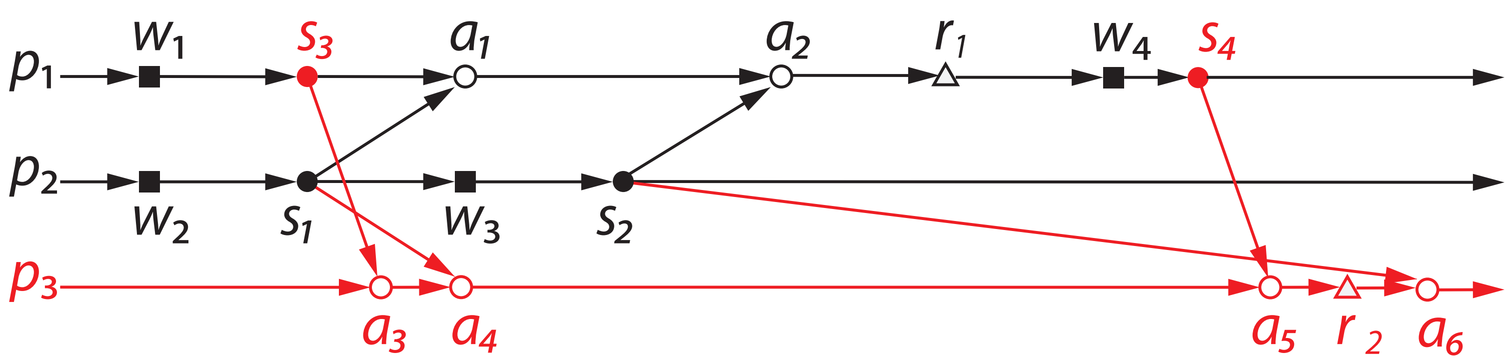
Causal consistency requires that if $w_3 \to w_4$, $w_3 \to a_6$, and $w_4 \to a_5$, then $a_5 \not\to a_6$.

This requirement is violated by the above picture.

$$\begin{tikzcd} w_3 \arrow[r] \arrow[d] & w_4 \arrow[d] \\ a_6 & a_5 \arrow[l, "/" marking] \end{tikzcd}$$

Schwarz and Mattern (1994) “Detecting causal relationships in distributed computations: in search of the holy grail.” *Distributed Computing*.

L:ingua Franca is Causally Consistency by Default

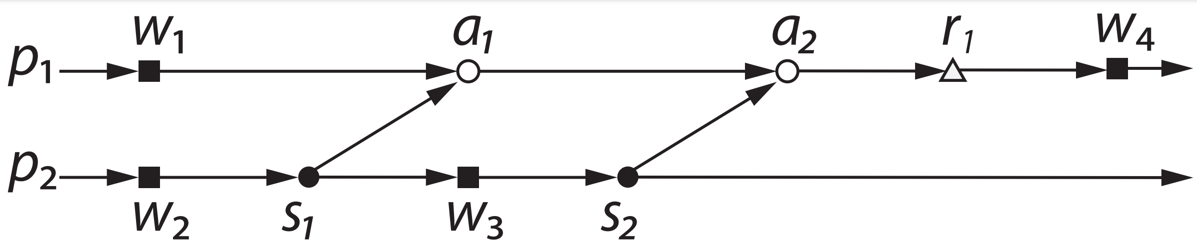


LF semantics: If a reaction r_1 can have an effect (state update or output) that influences what reaction r_2 sees in any way, then execution of r_1 must precede execution of r_2.

Note: This guarantee does not apply to side-effects!

Note: Adding **after** delays can explicitly reverse the ordering.

Recall Terminology



Event types:

- w_x : **Merge** a value with the local replica of x.
- r_x : **Read** the value of the local replica of x.
- s_x : **Send** the value of the local replica.
- a_x : **Accept** a new value for x and merge.

Inconsistency

Definition

For each write event on process j with tag g_j, let g_i be the tag of the corresponding accept event on process i or ∞ if there is no corresponding accept event. The **inconsistency** $\bar{C}_{ij} \in \mathbb{I}$ from j to i is defined to be

$$\bar{C}_{ij} = \max(\mathcal{T}(g_i) - \mathcal{T}(g_j)), \tag{1}$$

where the maximization is over all write events on process j. If there are no write events on j, then we define $\bar{C}_{ij} = 0$.

Time stamp of the tag.

Set of intervals.

Strong Consistency

A strongly consistent system is one where

$$\bar{c}_{ij} = 0$$

for all i, j.

In Lingua Franca, these means there are no after delays on connections between components.

Unavailability

Definition

For each read event on process i, let g_i be its tag and T_i be the physical time at which it is processed. The **unavailability** $\bar{A}_i \in \mathbb{I}$ at process i is defined to be

$$\bar{A}_i = \max(T_i - \mathcal{T}(g_i)), \tag{2}$$

where the maximization is over all read events on process i that are triggered by user requests. If there are no such read events on process i, then $\bar{A}_i = 0$.

Processing Offset

Definition

For process i, the **processing offset** $O_i \in \mathbb{I}$ is

$$O_i = \max(T_i - \mathcal{T}(g_i)) \tag{3}$$

where T_i and g_i are the physical time and tag, respectively, of a write event on process i that is triggered by a local external input (and hence assigned a timestamp drawn from the local clock). The maximization is over all such write events in process i. If there are no such write events, then $O_i = 0$.

Apparent Latency

Definition

Let g_j be the tag of a write event in process j that is triggered by an external input at j (so $\mathcal{T}(g_j)$ is the physical time of that external input). Let T_i be the physical time of the corresponding accept event in process i (or ∞ if there is no such event). (If $i = j$, we assume T_i is the same as the physical time of the write event.) The **apparent latency** or just **latency** $\mathcal{L}_{ij} \in \mathbb{I}$ for communication from j to i is

$$\mathcal{L}_{ij} = \max(T_i - \mathcal{T}(g_j)), \tag{4}$$

where maximization is over all such write events in process j. If there are no such write events, then $\mathcal{L}_{ij} = 0$.

More on Apparent Latency

The apparent latency is a sum of four components,

$$\mathcal{L}_{ij} = O_j + X_{ij} + L_{ij} + E_{ij}, \tag{5}$$

where X_{ij} is **execution time** overhead at node j for sending a message to node i, L_{ij} is the **network latency** from j to i, and E_{ij} is the **clock synchronization error**. The three latter quantities are indistinguishable and always appear summed together, so there is no point in breaking apparent latency down in this way. Moreover, these latter three quantities would have to be measured with some physical clock, and it is not clear what clock to use. The apparent latency requires no problematic measurement since it explicitly refers to local clocks and tags.

CAL Theorem for Strongly Consistent Systems

The unavailability at process *i* for a strongly consistent system is

$$\bar{A}_i = \max_{j \in N} \max(\mathcal{L}_{ij}, o_i) = \max(o_i, \max_{j \in N} \mathcal{L}_{ij}). \tag{6}$$

When network latency, clock synchronization error, or execution time increase, the apparent latency increases, and so does the unavailability. "Network partitioning" means that $\mathcal{L}_{ij}$ diverges and the system becomes unavailable at process *i*, $\bar{A}_i = \infty$. When enforcing strong consistency, therefore, network partitioning implies unavailability, as expected from the CAP theorem.

The CAL Theorem for Arbitrary Consistency

Theorem

Given a trace, the unavailability at process i is, in the worst case,

$$\bar{A}_i = \max\left(O_i, \max_{j \in N}(\mathcal{L}_{ij} - \bar{C}_{ij})\right), \tag{7}$$

where O_i is the processing offset, $\mathcal{L}_{ij}$ is apparent latency (which includes O_j), and $\bar{C}_{ij}$ is the inconsistency.

Max-Plus Algebra

Operators:

$$a \oplus b = \max(a, b)$$

$$a \otimes b = a + B - side$$

Algebra properties:

- associativity:
  $(a \oplus b) \oplus c = a \oplus (b \oplus c)$
  $(a \otimes b) \otimes c = a \otimes (b \otimes c)$
- commutativity:
  $a \oplus b = b \oplus a$
  (note: $\otimes$ is not commutative for matrix ops)
- distributivity:
  $(a \oplus b) \otimes b = (a \otimes c) \oplus (b \otimes c)$

Connection Matrix

Let N be the number of nodes, and define an $N \times N$ matrix Γ such that its elements are given by

$$\Gamma_{ij} = \mathcal{L}_{ij} - \bar{C}_{ij} - O_j. \tag{8}$$

That is, the i, j-th entry in the matrix is an assumed bound on $X_{ij} + L_{ij} + E_{ij}$ (execution time, network latency, and clock synchronization error), adjusted downwards by the specified tolerance $\bar{C}_{ij}$ for inconsistency.

The CAL Theorem in Max-Plus

Let $\boldsymbol{A}$ be a column vector with elements equal to the unavailabilities $\bar{A}_i$, and $\boldsymbol{O}$ be a column vector with elements equal to the processing offsets O_i. Then the CAL theorem can be written as

$$\boldsymbol{A} = \boldsymbol{O} \oplus \Gamma \boldsymbol{O}, \tag{9}$$

where the matrix multiplication is in the max-plus algebra. This can be rewritten as

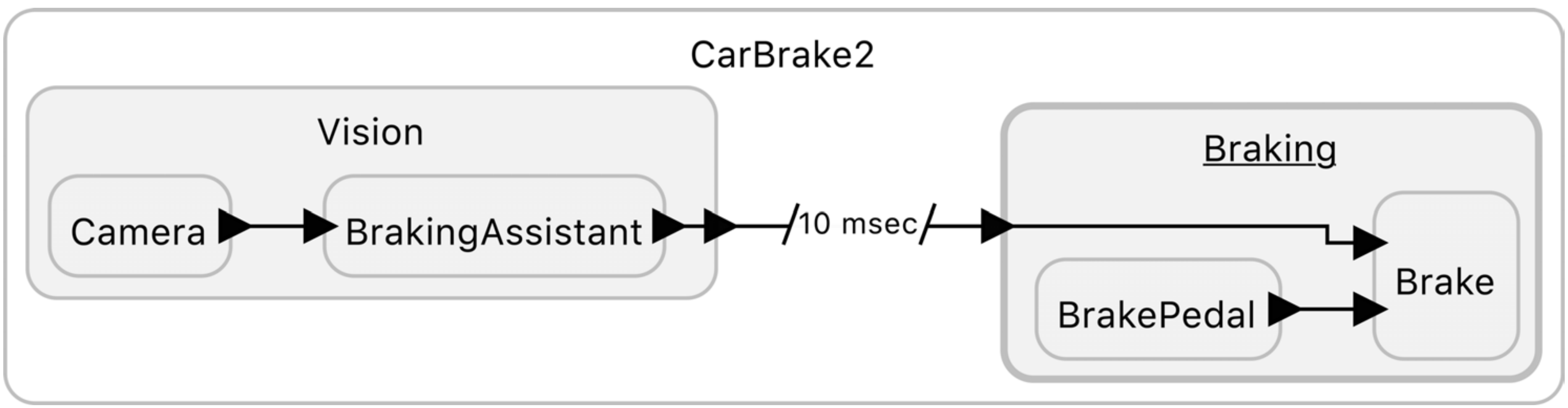
$$\boldsymbol{A} = (\boldsymbol{I} \oplus \Gamma) \boldsymbol{O}, \tag{10}$$

where $\boldsymbol{I}$ is the identity matrix in max-plus, which has zeros along the diagonal and $-\infty$ everywhere else.

Lingua Franca enables explicit tradeoffs between Consistency and Availability

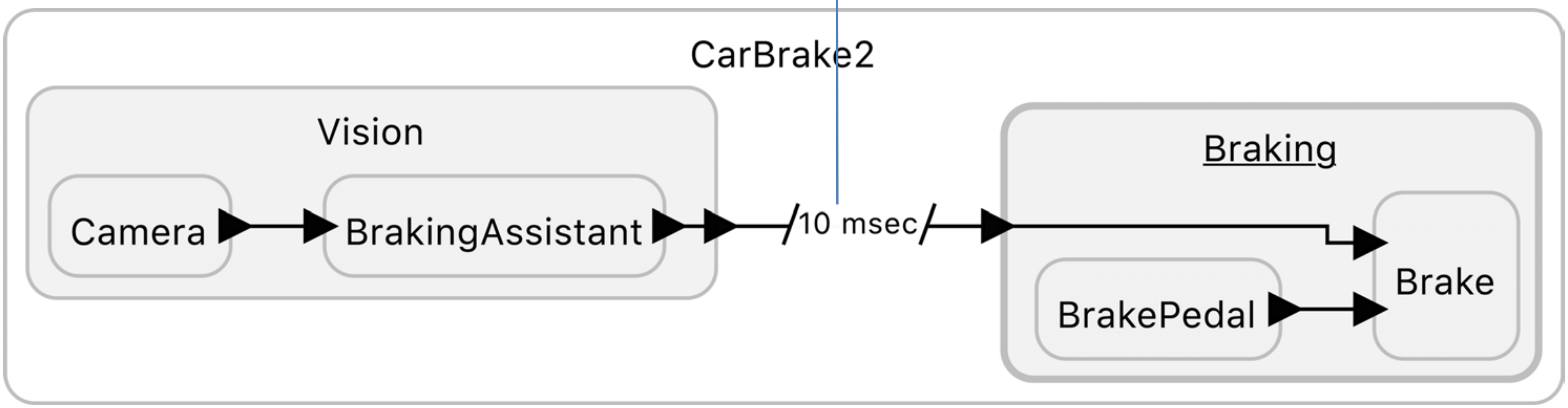
- Specify your availability requirement (deadline)
- State your requirements of the network (latency)
- Specify a tolerance for inconsistency (logical delay)

E.g., 10 msec tolerance for logical delay implies that if network latencies are less than 10 msec, availability is instantaneous:



What if the Execution Time Bound is Violated?

If communication latency exceeds 10 msec…



choice

Sacrifice availability

Sacrifice consistency

Fault Handling in Lingua Franca

Led by Soroush Bateni (UT Dallas, UC Berkeley)

Sacrifice availability

Centralized coordination:

- Use centralized coordination
- Provide deadline miss handlers.

Sacrifice consistency

Decentralized coordination:

- Use decentralized coordination.
- Provide safe-to-process violation handlers.