Distributed Collaboration among Agents

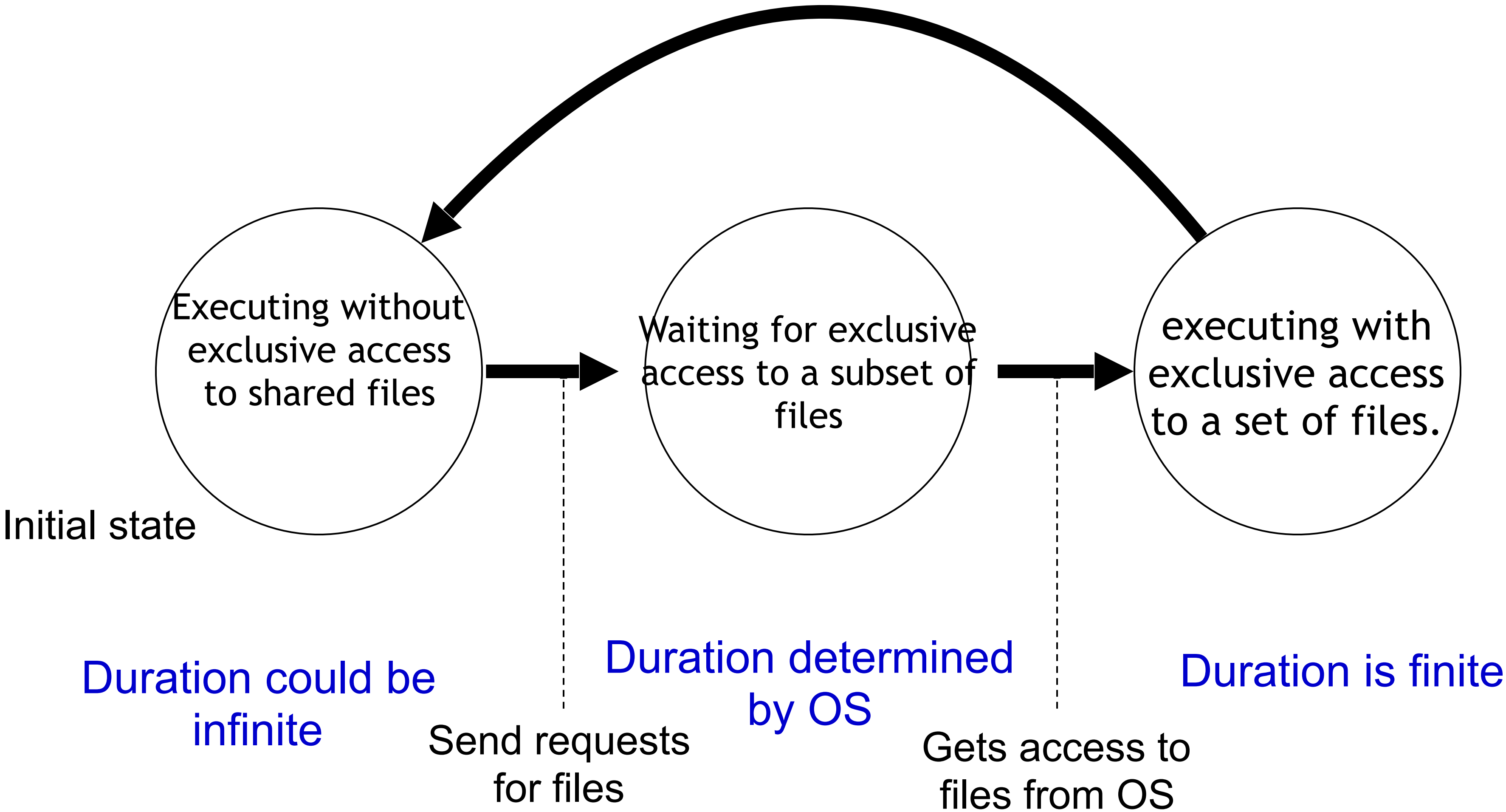
Agents need exclusive access to
a set of files

The Drinking Philosophers Problem

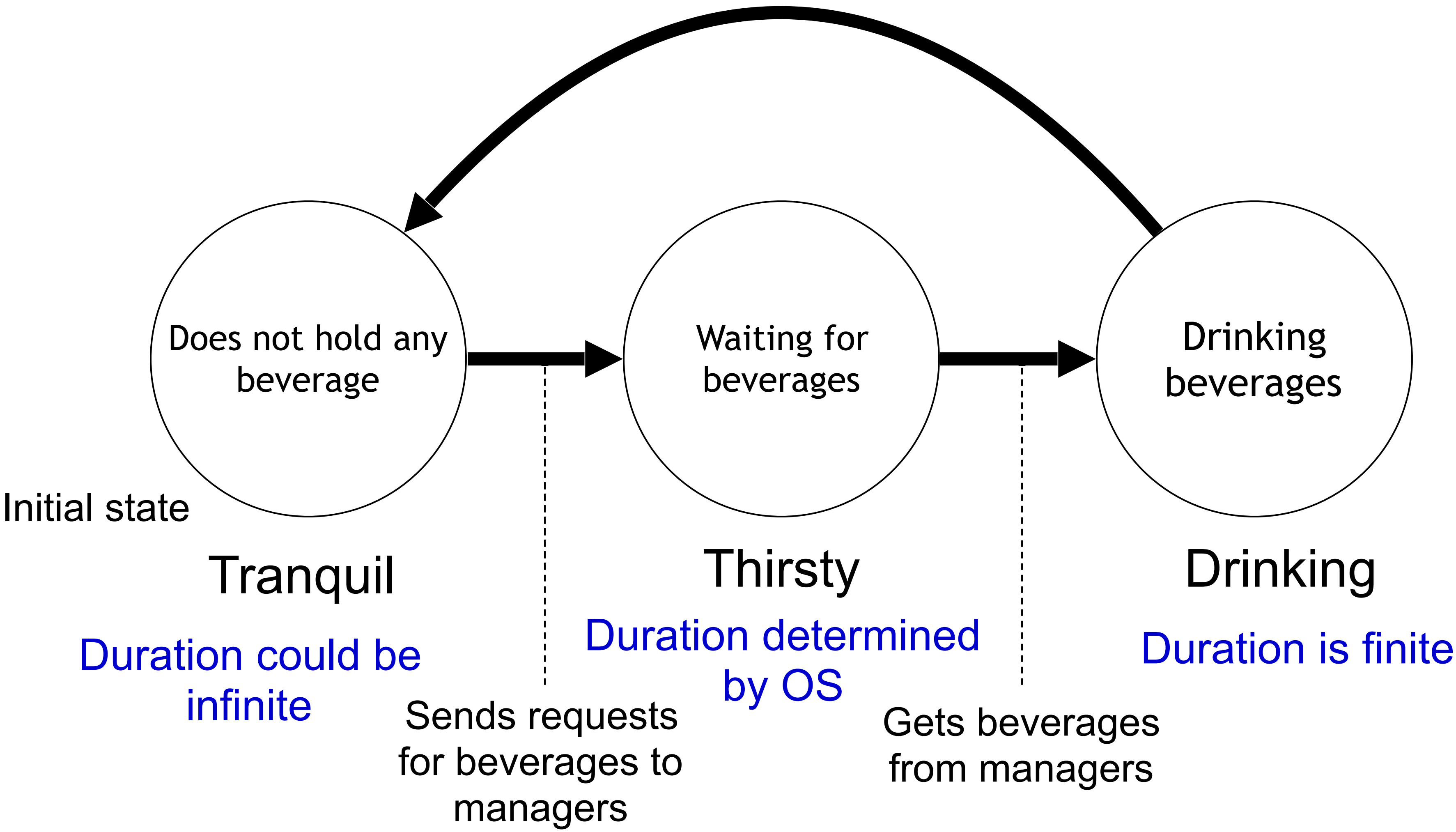
Key ideas of drinking philosophers algorithm

1. **Conflict resolution in distributed systems**.
2. **Priority among agents in conflict.** Some agents win and others lose. Fair winning: every agent that wants to win gets to win *eventually*.
3. **Tokens.** An agent that holds a token knows that other agents don't hold the same token.
4. **Dynamic Data Structures**: Priorities based on timestamps and agent ids.

Client Life Cycle: Same as for mutex



Client Life Cycle: Similar to Dining Philosophers

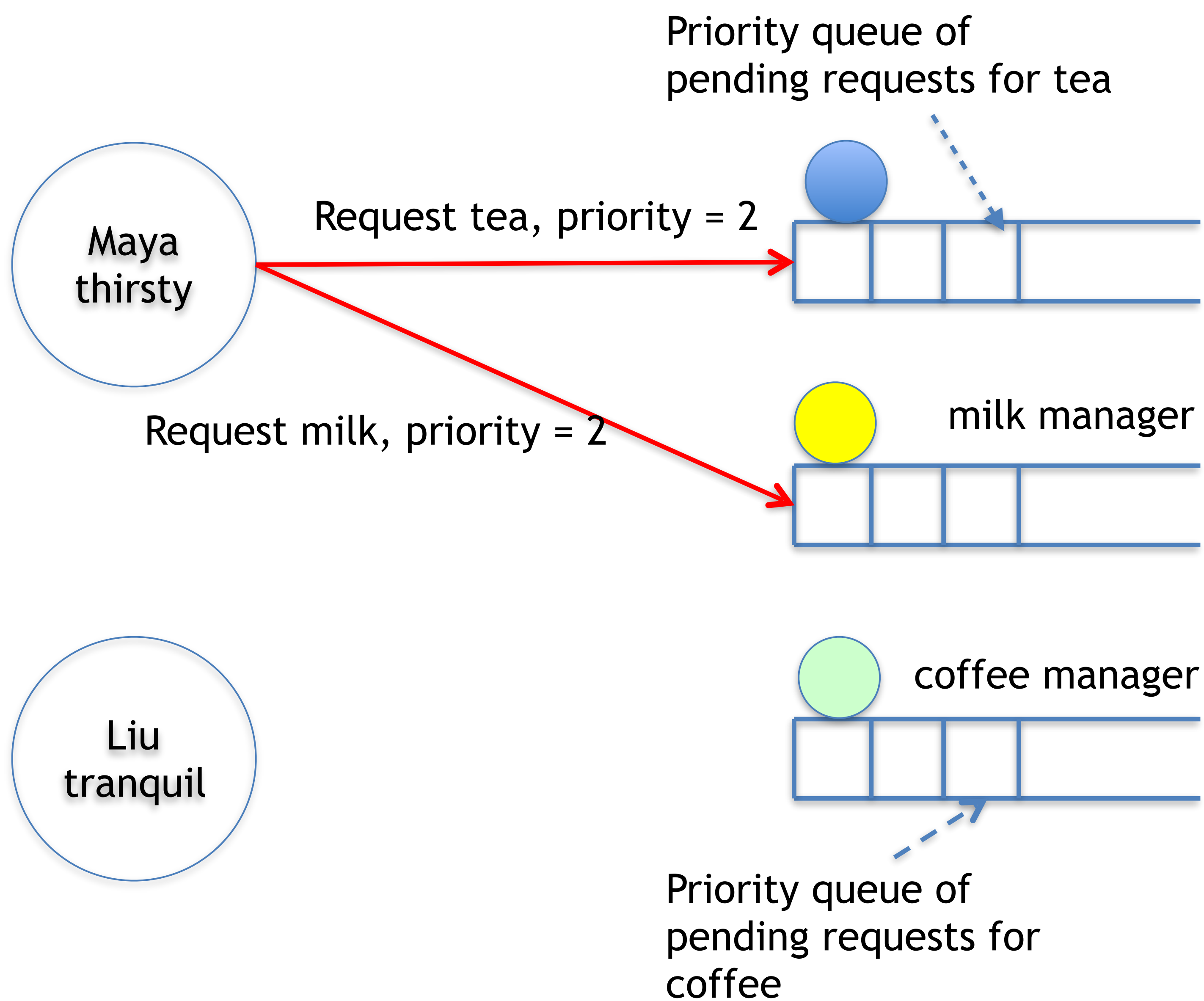


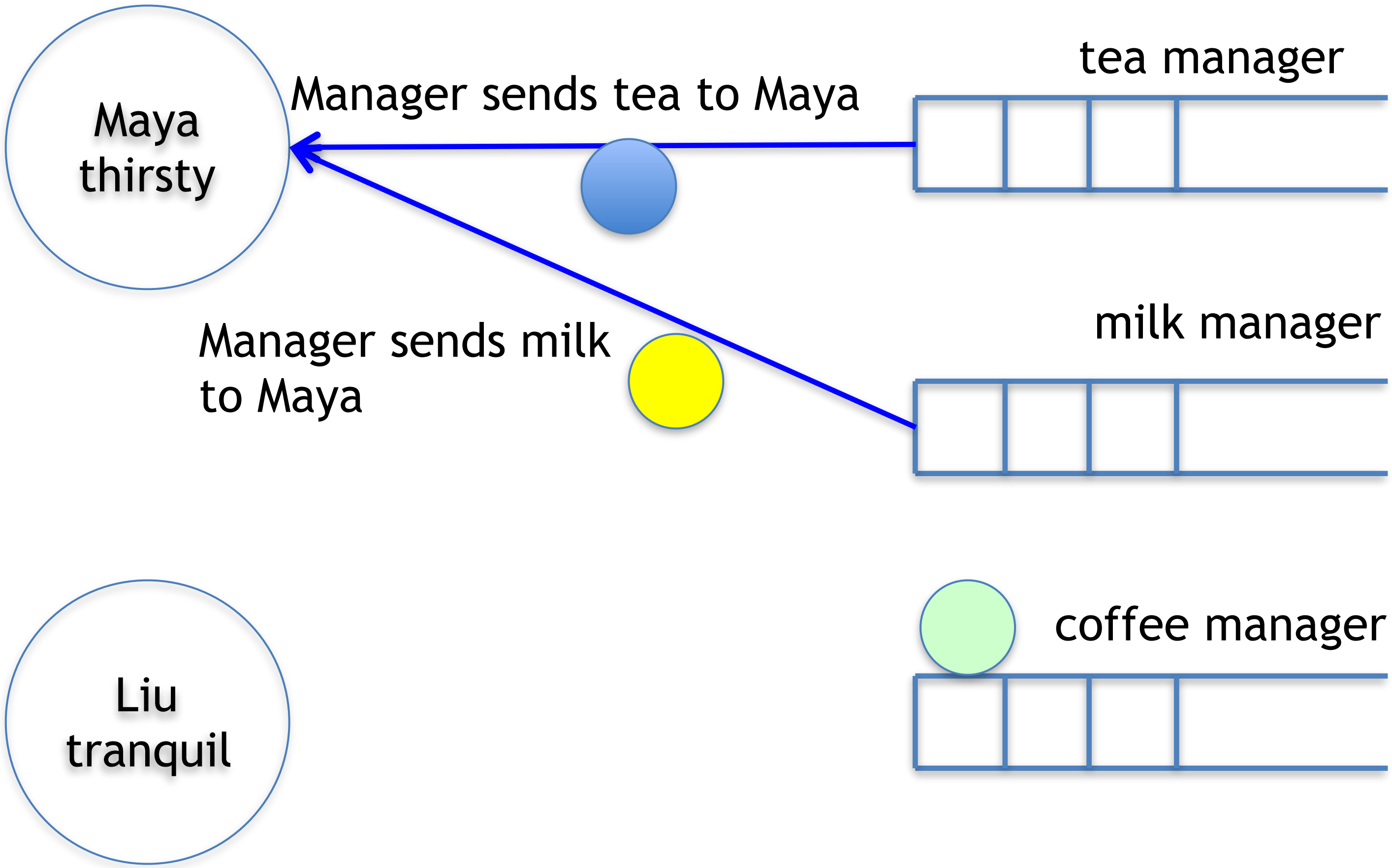
Example:

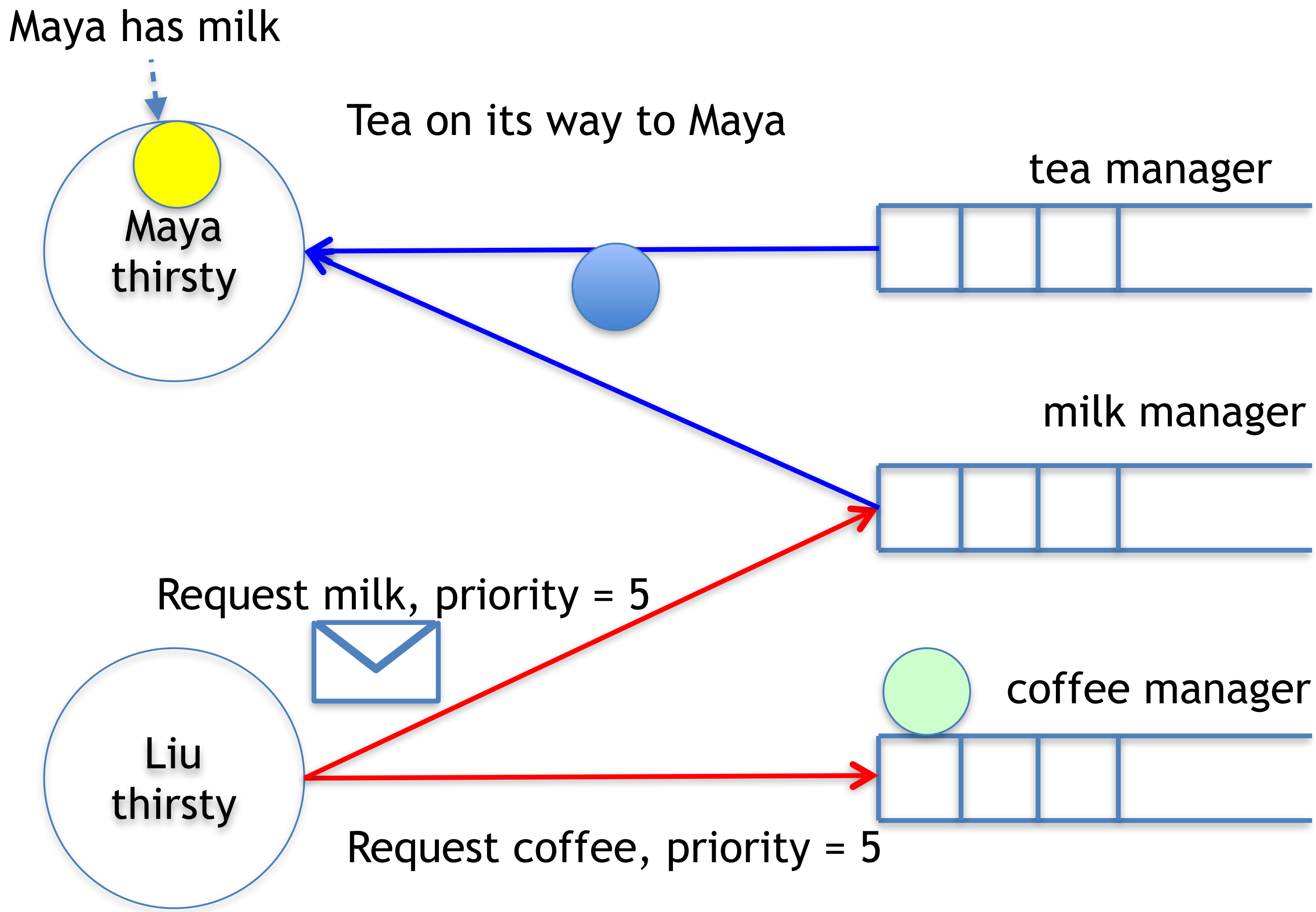
Agents: Maya and Liu
Maya has priority 2
Liu has higher priority: 5

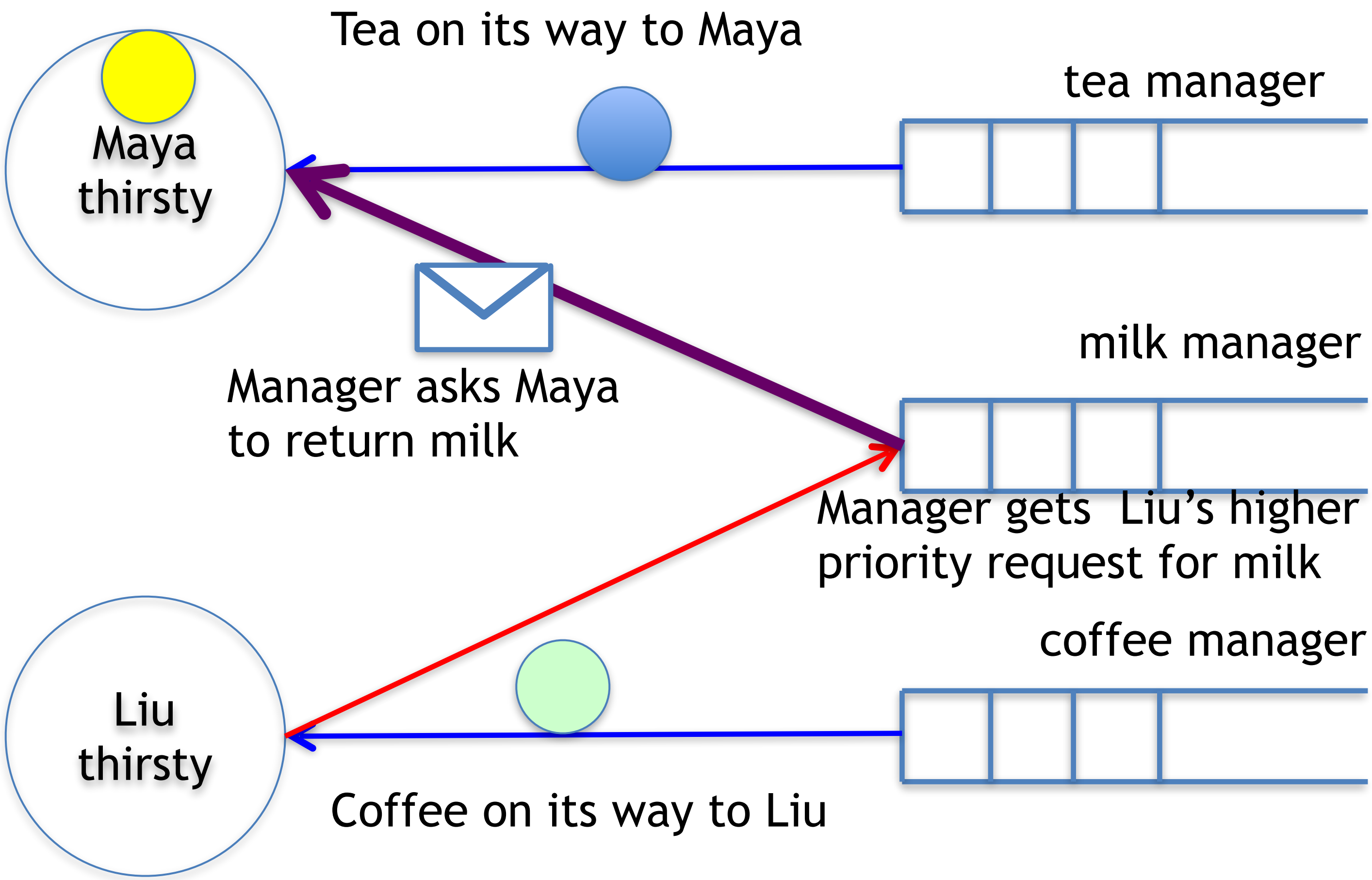
Resources: tea, milk, coffee

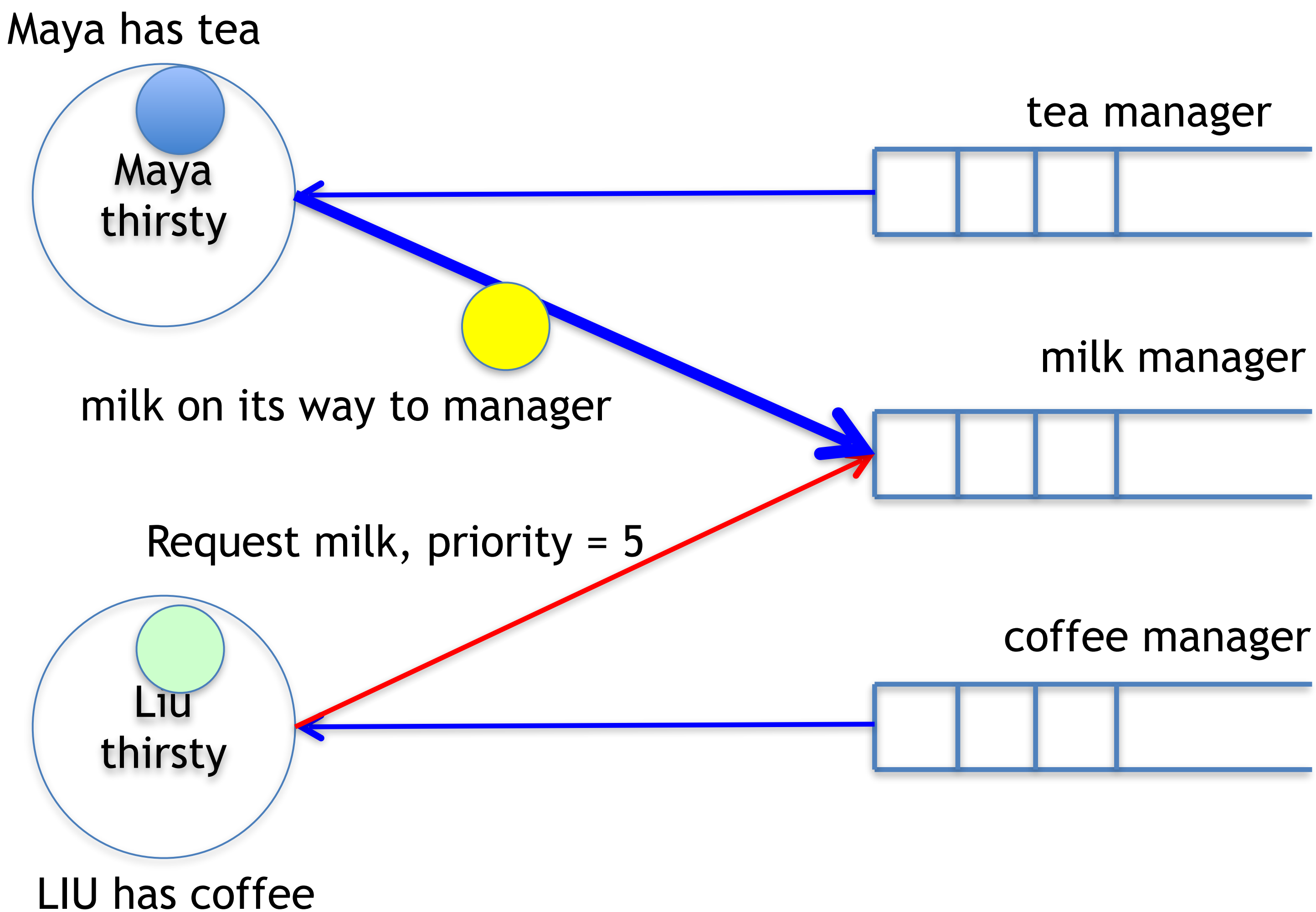
Agents may become thirsty for any (nonempty) set of resources.
e.g. Maya becomes thirsty for milk and tea;
after she gets milk and tea, she drinks it;
becomes tranquil;
becomes thirsty for coffee;
after she gets coffee she drinks it;
becomes tranquil;
becomes thirsty for tea and coffee….

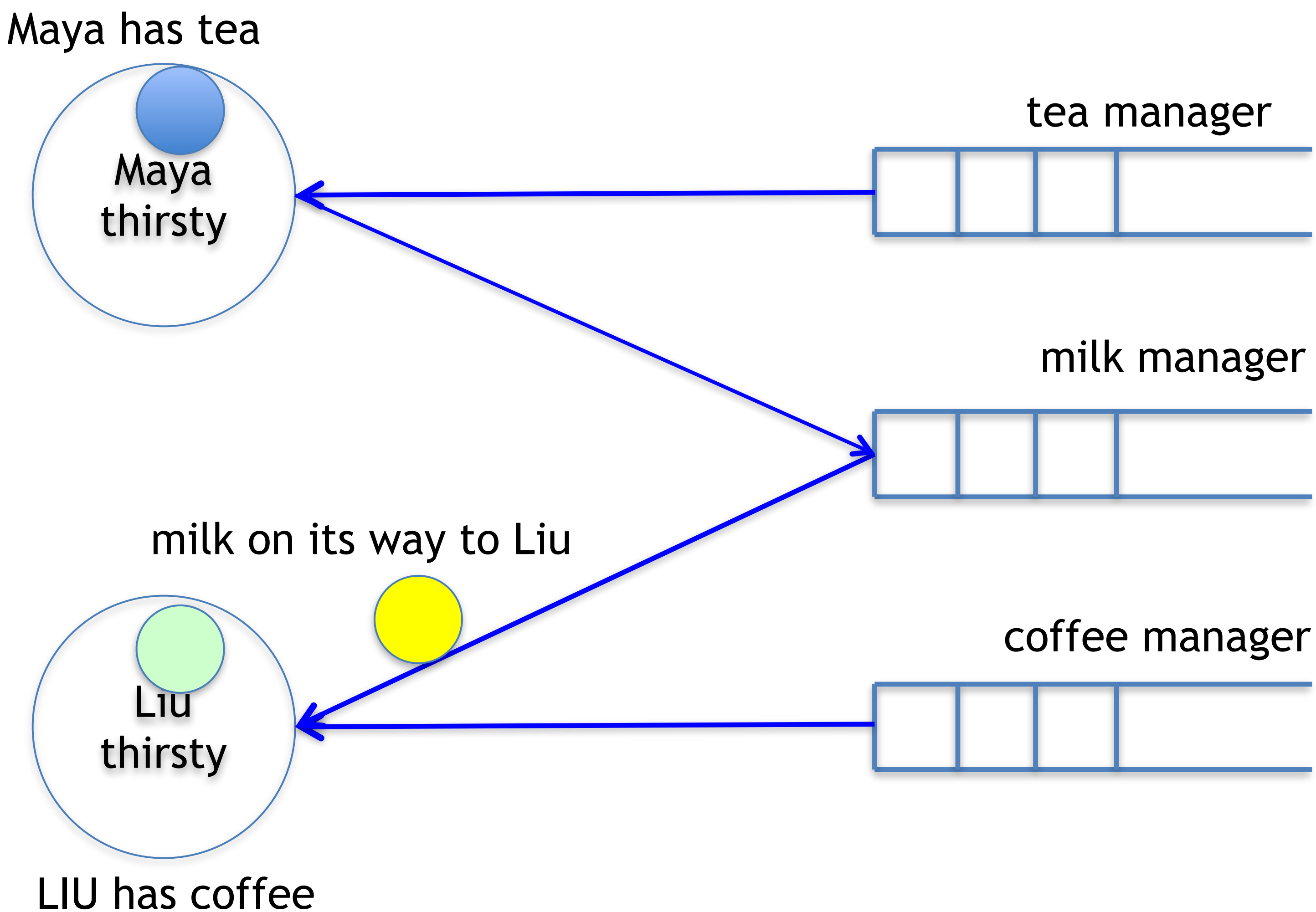

Priority queue of
pending requests for tea
Maya
thirsty
Request tea, priority = 2
milk manager
Request milk, priority = 2
Liu
tranquil
coffee manager
Priority queue of
pending requests for
coffee

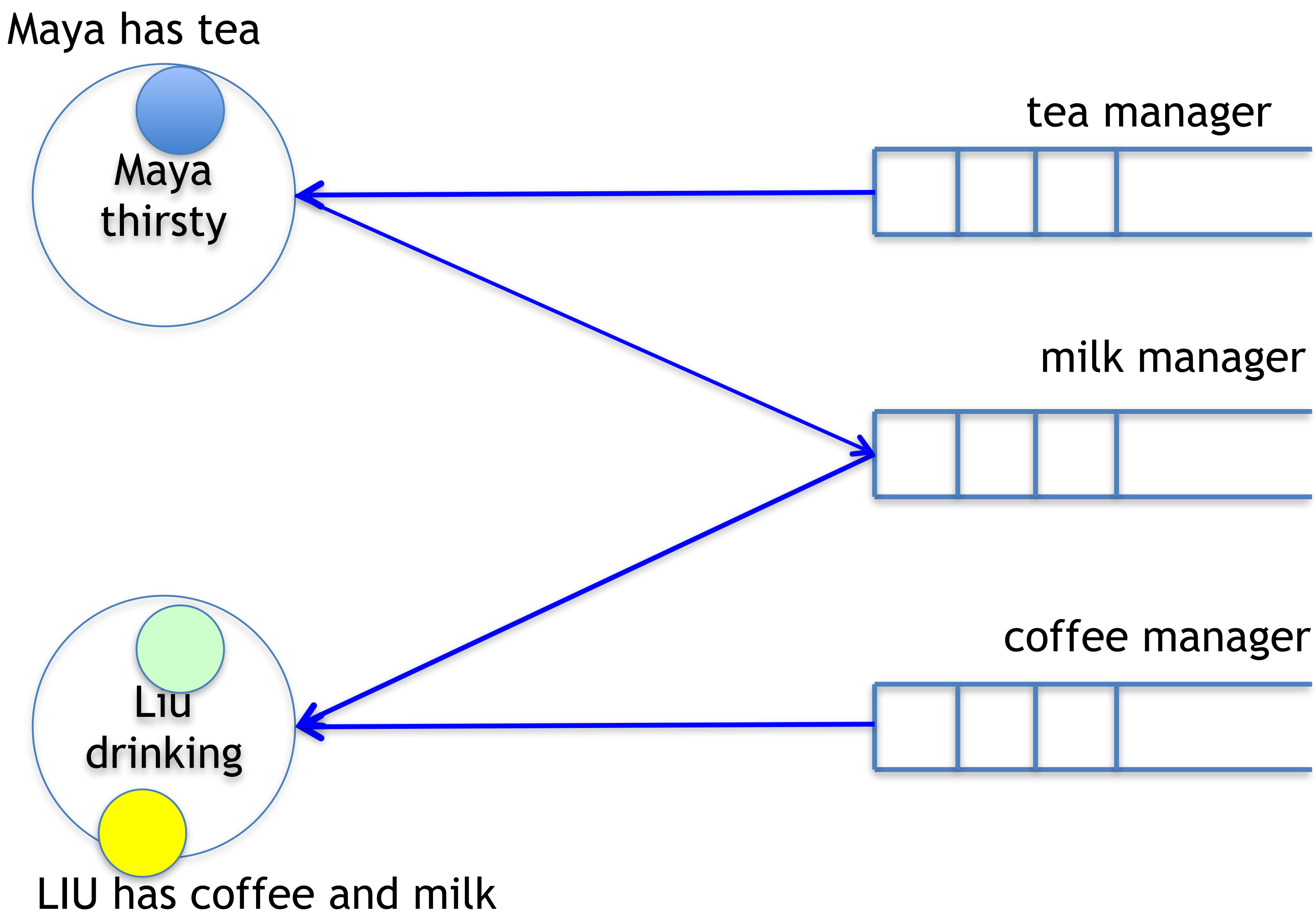

Maya
thirsty
Manager sends tea to Maya
tea manager
Manager sends milk
to Maya
milk manager
coffee manager
Liu
tranquil

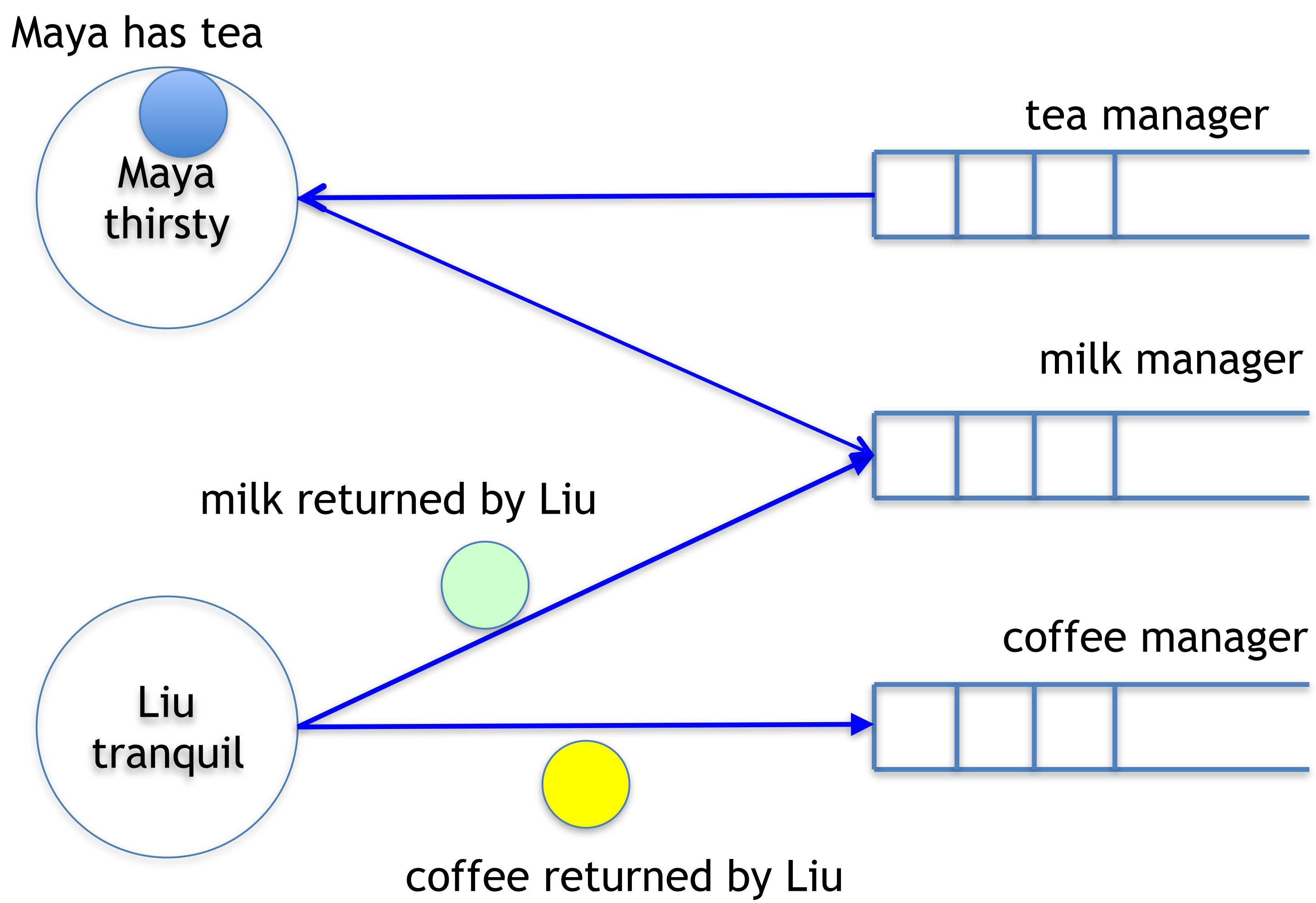

Maya has milk
Tea on its way to Maya
tea manager
Maya
thirsty
milk manager
Request milk, priority = 5
coffee manager
Liu
thirsty
Request coffee, priority = 5

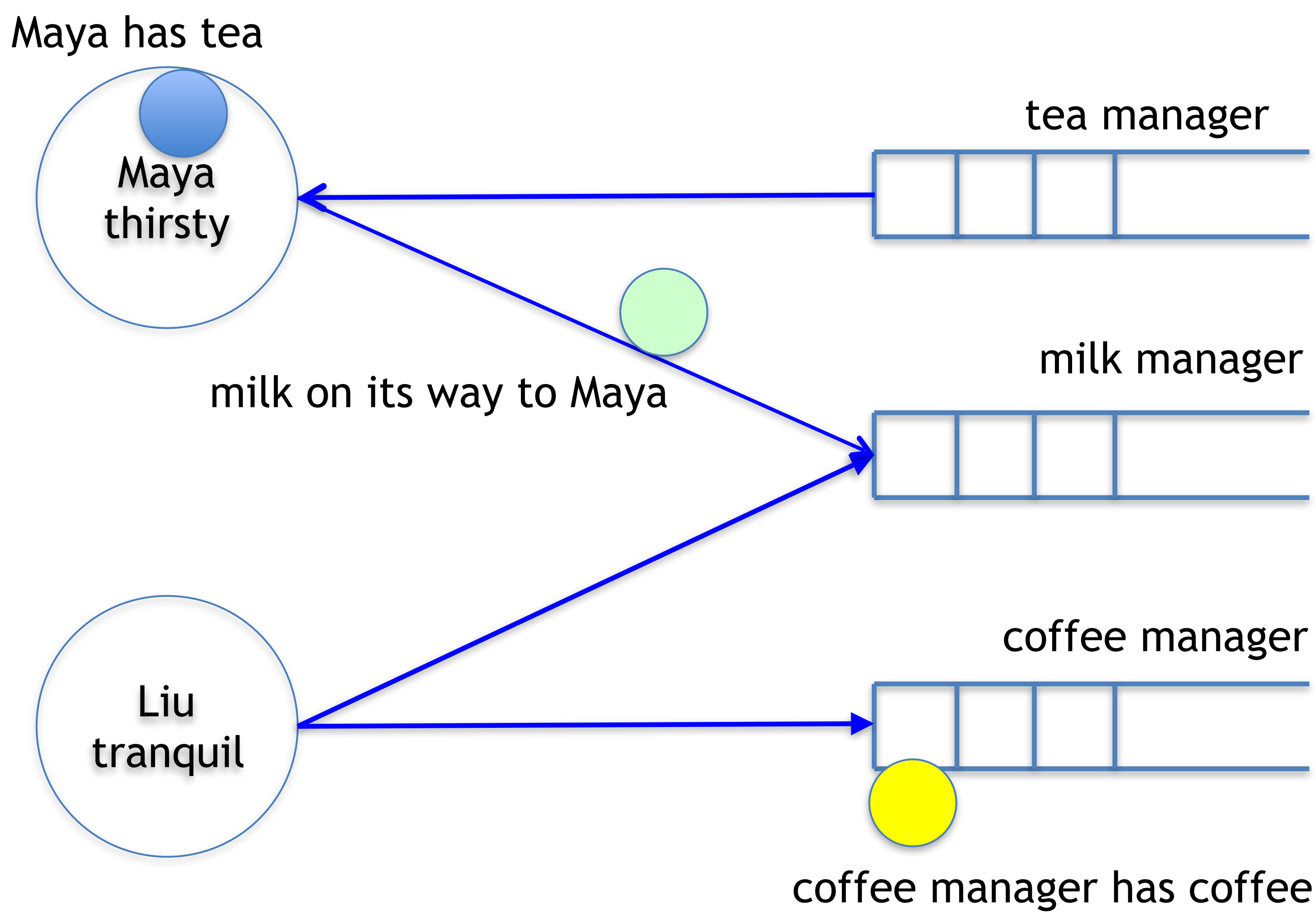

Tea on its way to Maya
tea manager
Maya
thirsty
milk manager
Manager asks Maya
to return milk
Manager gets Liu's higher
priority request for milk
coffee manager
Liu
thirsty
Coffee on its way to Liu


Maya has tea
Maya
thirsty
tea manager
milk on its way to manager
milk manager
Request milk, priority = 5
coffee manager
Liu
thirsty
LIU has coffee


Maya has tea
Maya
thirsty
tea manager
milk manager
milk on its way to Liu
coffee manager
Liu
thirsty
LIU has coffee


Maya has tea
Maya
thirsty
tea manager
milk manager
coffee manager
Liu
drinking
LIU has coffee and milk


Maya has tea
Maya
thirsty
tea manager
milk manager
milk returned by Liu
Liu
tranquil
coffee manager
coffee returned by Liu


Maya has tea
Maya
thirsty
tea manager
milk on its way to Maya
milk manager
Liu
tranquil
coffee manager
coffee manager has coffee

Assume each agent has a unique id: a number
Give priority to the agent with the highest id.

Will that work?

Assume each agent has a unique id: a number
Give priority to the agent with the highest id.

Will that work?

No, because an agent with the highest id can win conflicts forever, and lower id agents may remain thirsty forever.

What to do?

Assume each agent has a unique id: a number
Give priority to the agent with the highest id.

Will that work?

No, because an agent with a high id can win conflicts forever.

What to do?

Priorities for agents that lose the conflict must increase with respect to the winner. *How*?

Assume each agent has a unique id: a number
Give priority to the agent with the highest id.

Will that work?

No, because an agent with a high id can win conflicts forever.

What to do?

Priorities for agents that lose a conflict must increase with respect to the winner. How?

Priority is (timestamp, agent_id) where timestamp is the local clock time when the agent requests resource.

Proof of Correctness

Safety: A resource is held by at most one agent at a time.

Proof: tokens are not created or destroyed or changed.

for each color X:
always(system has exactly one token of color X)

Proof that a request with timestamp T gets its resources eventually.

Part 1: Eventually all agents' clocks exceed T

Part 2: After all agents' clocks exceed T the number of pending requests with timestamps T or less decreases to 0.

Proof that a request with timestamp T gets its resources eventually.

Part 1: A state is reached in which all agents' clocks exceed T.

Why?

Part 1: A state is reached in which all agents' clocks exceed T.

Why?

(*Note: This is a proof about properties of clocks and is not specific to the drinking philosophers problem.*)

Because each agent's clock ticks forward by at least 1 and agent clocks never go backwards.

For any T: local clock times of each agent *i* ticks forward by at least 1 (from the specification of local clocks)

For all i, all k: t[i] = k   leads-to   t[i] >= k+1

For any T: local clock times of each agent *i* ticks forward by at least 1 (from the specification of local clocks)

For all i, all k: t[i] = k leads-to t[i] >= k+1

Transitivity:

For all i, all k: t[i] = k leads-to t[i] > T

For any T: local clock times of each agent *i* ticks forward by at least 1 (from the specification of local clocks)

For all i, all k: t[i] = k leads-to t[i] >= k+1

Transitivity:

For all i, all k: t[i] = k leads-to t[i] > T

Disjunction:

Eventually local clock times of each agent exceeds T

For all i: true leads-to t[i] > T

Eventually local clock times of each agent exceeds T

For all i: *true* *leads-to* *t[i] > T*

Clock times never decrease

For all i: *stable(t[i] > T)*

Eventually a state is reached in which each agent's clock times exceeds T and remains in excess of T.

For all i: *true* *leads-to* *always(t[i] > T)*

Eventually local clock times of each agent exceeds T

For all i: true leads-to t[i] > T

Clock times never decrease

For all i: stable(t[i] > T)

Eventually a state is reached in which each agent's clock times exceeds T and remains in excess of T.

For all i: true leads-to always(t[i] > T)

Because (P leads-to always(Q)) AND (P leads-to always(Q'))

IMPLIES

(P leads-to always(Q AND Q'))

we get: *true leads-to always(for all i: t[i] > T)*

Part 2:
Let P be the predicate

P: all clocks exceed T

Let M be number of requests with timestamp less than T.
M is a variant function.

Prove for all $k > 0$:
(P AND $(M = k)$) leads-to (P AND $(M < k)$)

Proof: Pending request with lowest timestamp gets its resources.

Using local clocks in distributed conflict resolution

Part 1

Given each agent's local clock ticks forward prove that
for all T: eventually all agent's local clocks exceed T

Part 2

All agent's local clocks exceed T and
k > 0 pending requests with timestamp T or less

leads-to

All agent's local clocks exceed T and
fewer than k pending requests with timestamp T or less

Using local clocks in distributed conflict resolution

Part 1: eventually all agent's local clocks exceed T

Same proof can be used in most problems.

Part 2

Pending requests with timestamp T or less decreases

The proof varies from problem to problem.
The proof depends on how agents request resources.

Key ideas of drinking philosophers algorithm

1. **Conflict resolution in distributed systems**.
2. **Priority among agents in conflict.** Some agents win and others lose. Fair winning: every agent that wants to win gets to win *eventually*.
3. **Tokens.** An agent that holds a token knows that other agents don't hold the same token.
4. **Dynamic Data Structures**: Priorities based on timestamps and agent ids.

Another algorithm for distributed conflict resolution

Resources are ordered.

Assume that each resource (eg. file) has an integer id.

Each agent requests resource *i* only after it holds all resources that it needs with id greater than *i*.

Example: Suppose an agent needs beverages 10, 7, 3 to transition from tranquil to drinking (in drinking philosophers).

The agent first requests beverage 10. Only after it holds beverage 10 does it request beverage 7. Only after it holds beverages 10, 7 does it request beverage 3.

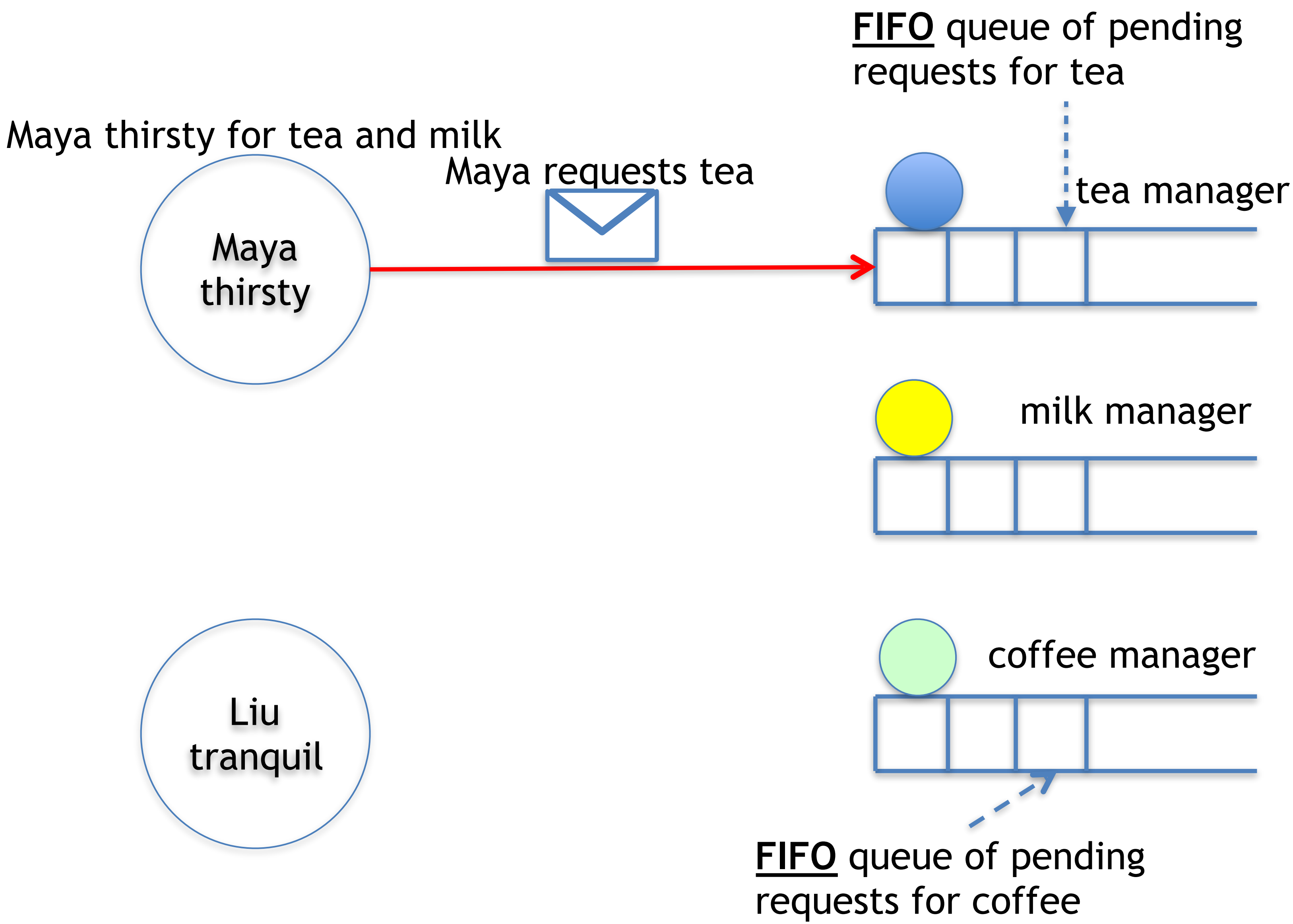
When it has all beverages it needs the agent drinks.

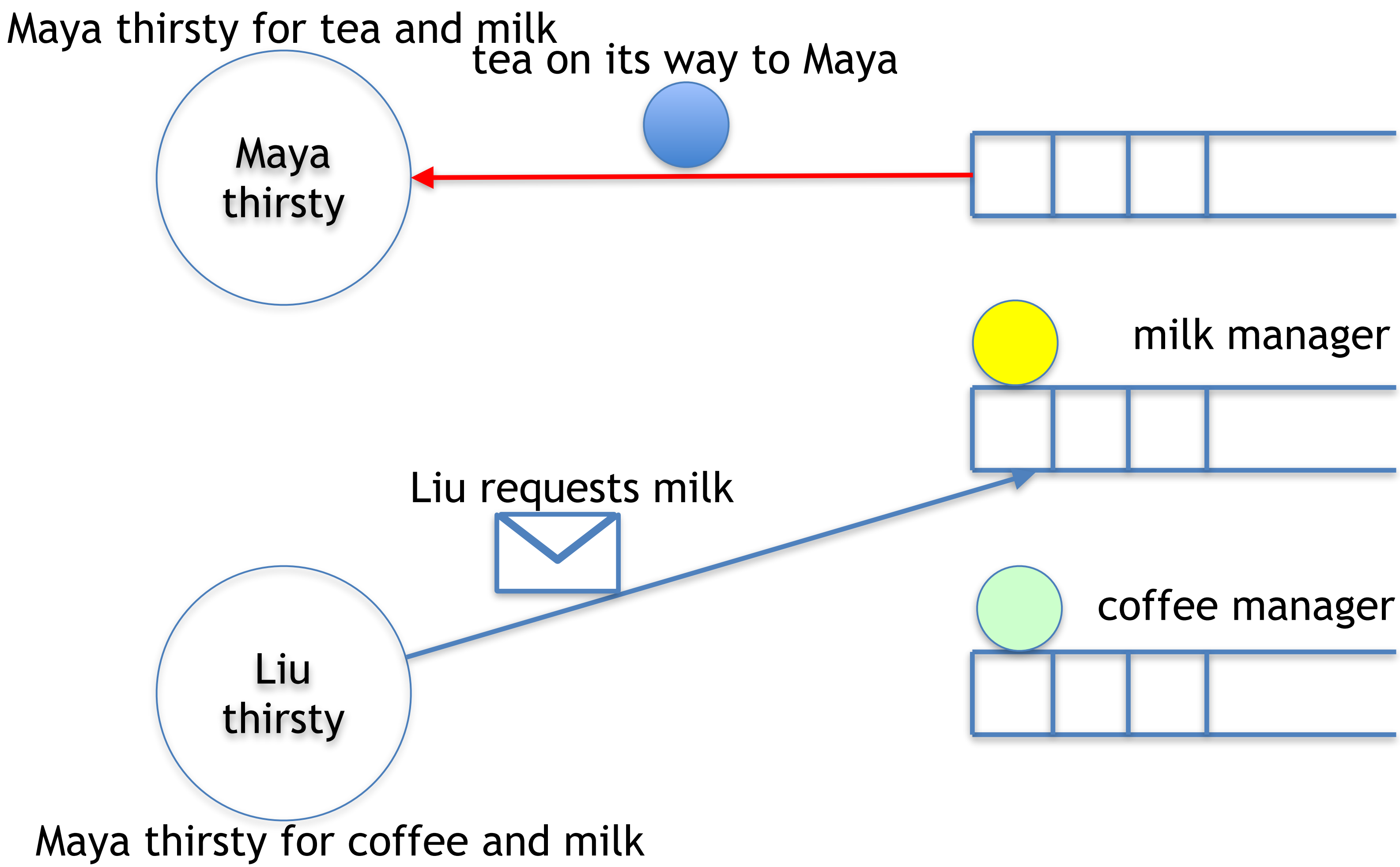
Example

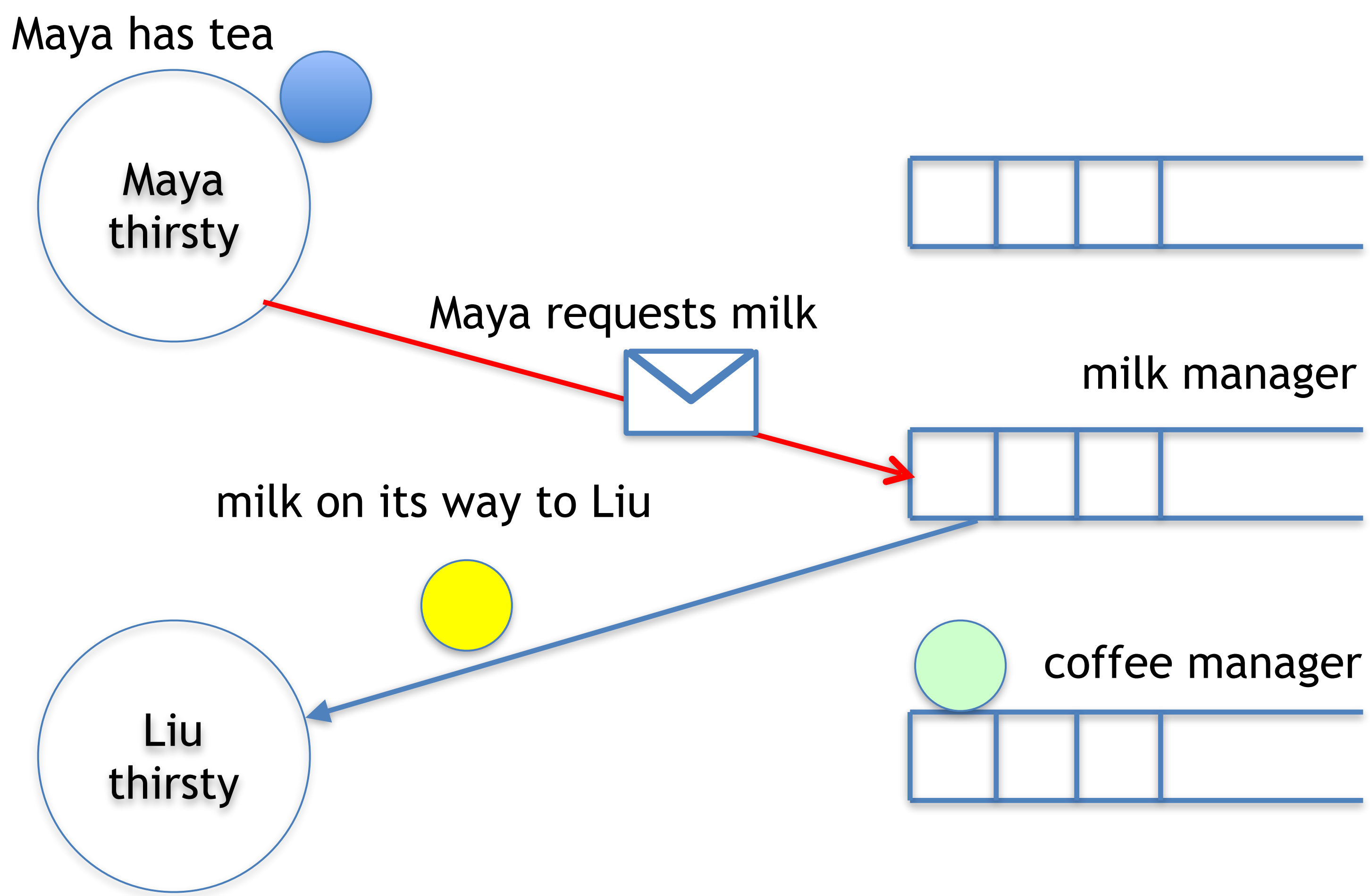
Order of resources: tea > milk > coffee

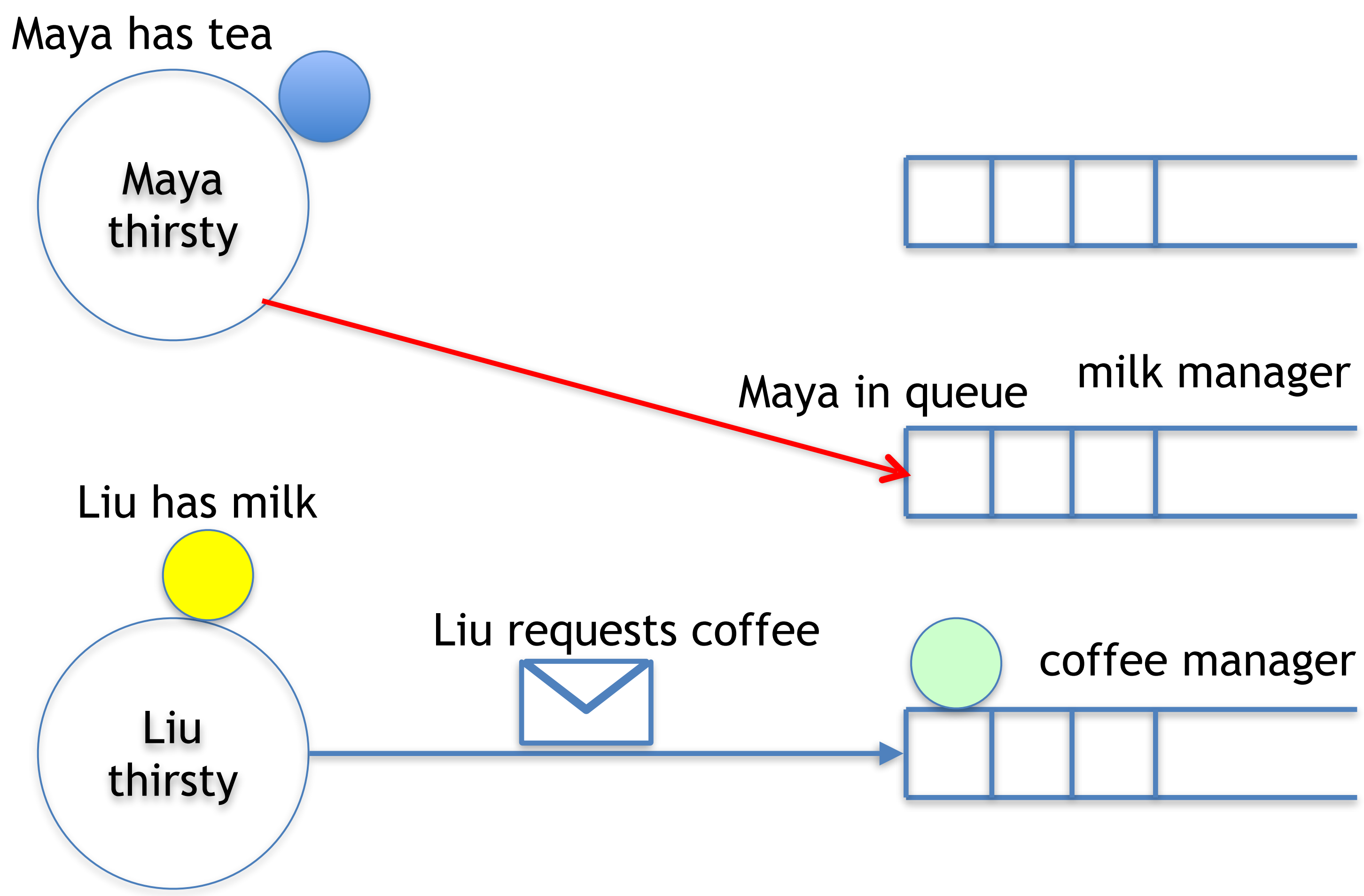
Maya thirsty for tea and milk
Liu thirsty for milk and coffee

Queues are First-In-First-Out (FIFO), not priority queues.

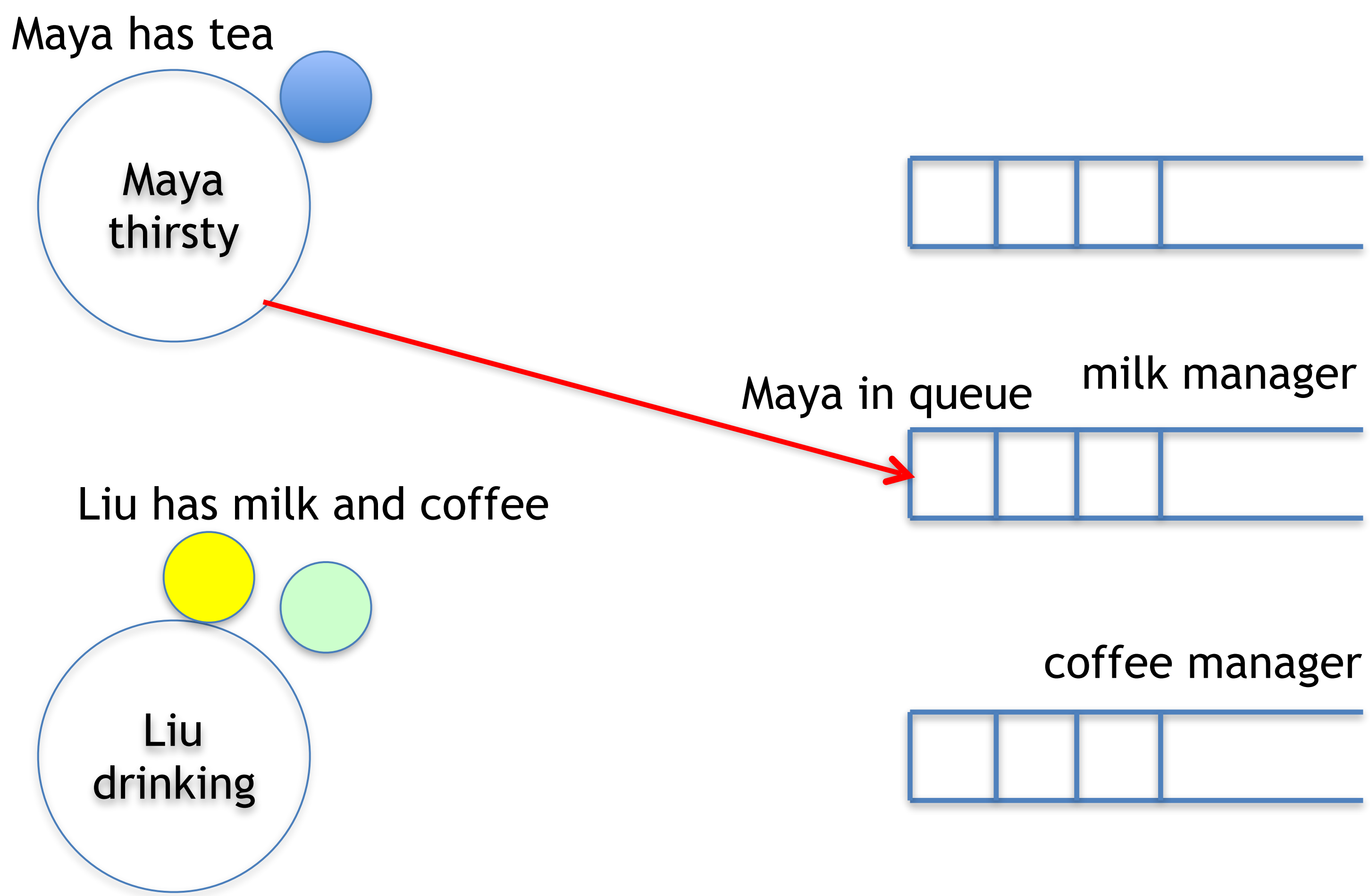

FIFO queue of pending requests for tea
Maya thirsty for tea and milk
Maya requests tea
tea manager
Maya thirsty
milk manager
coffee manager
Liu tranquil
FIFO queue of pending requests for coffee

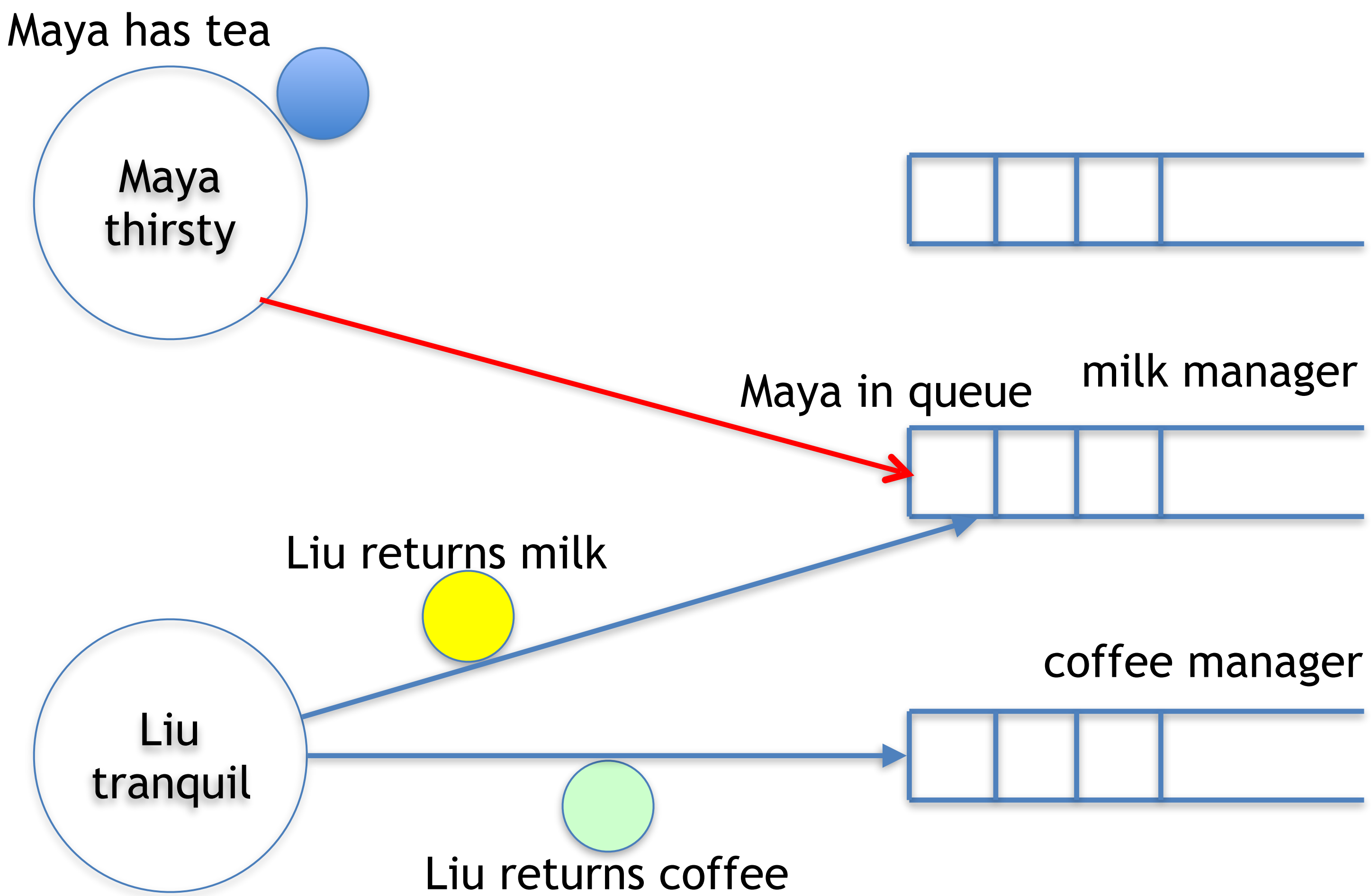

Maya thirsty for tea and milk
tea on its way to Maya
Maya
thirsty
milk manager
Liu requests milk
coffee manager
Liu
thirsty
Maya thirsty for coffee and milk

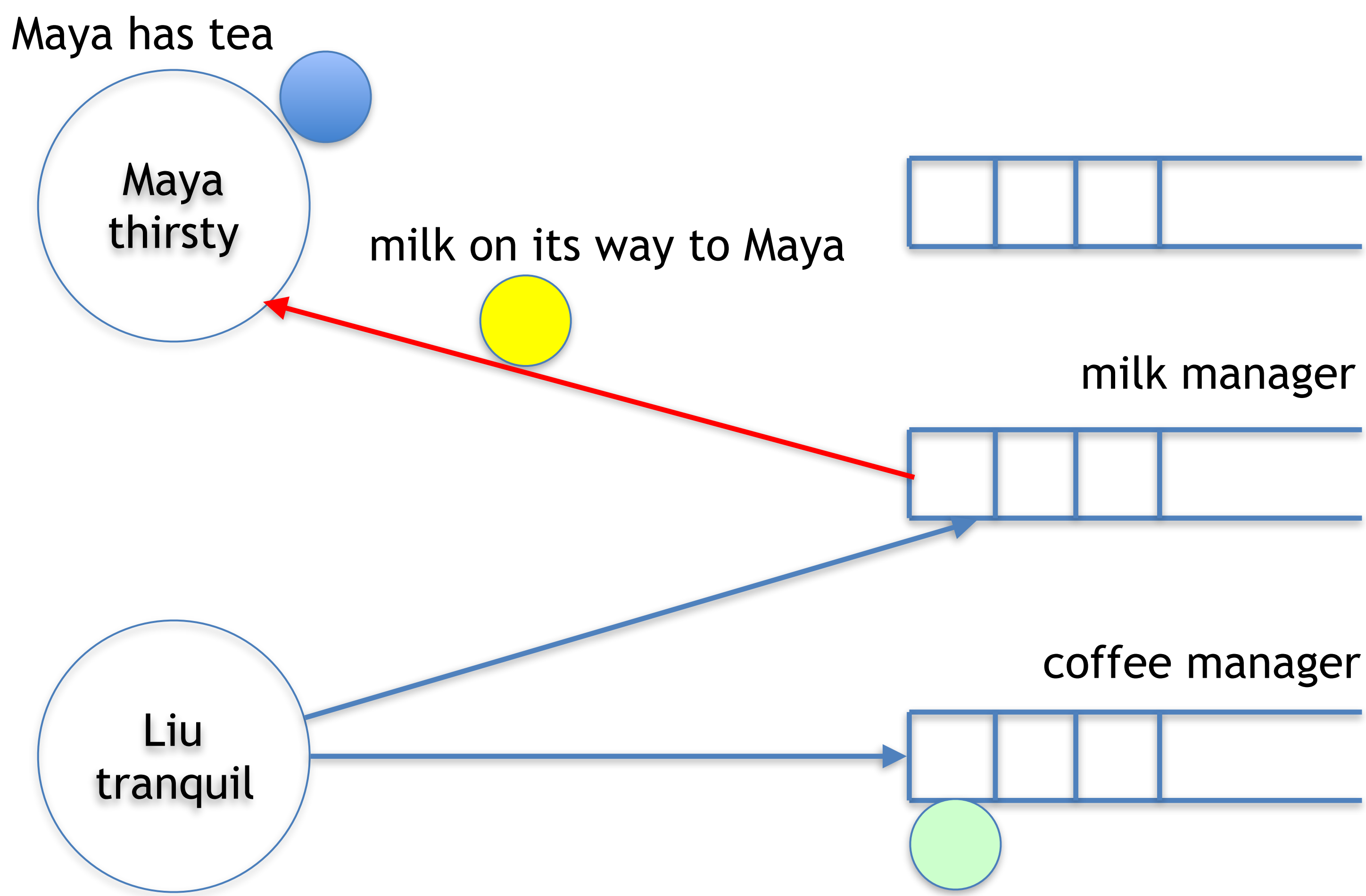

Maya has tea
Maya thirsty
Maya requests milk
milk manager
milk on its way to Liu
coffee manager
Liu thirsty



Maya has tea
Maya thirsty
Maya in queue
milk manager
Liu has milk
Liu requests coffee
coffee manager
Liu thirsty


Maya has tea
Maya
thirsty
Maya in queue
milk manager
Liu has milk and coffee
Liu
drinking
coffee manager


Maya has tea
Maya
thirsty
Maya in queue
milk manager
Liu returns milk
coffee manager
Liu
tranquil
Liu returns coffee


Maya has tea
Maya
thirsty
milk on its way to Maya
milk manager
coffee manager
Liu
tranquil

Proof of Correctness

Safety: A resource is held by at most one agent. (Straightforward)

Progress: By induction on resource id n

Induction hypothesis: All pending requests for a set R of resources where the lowest resource id in R is n are satisfied.

Base case: n = 0, were 0 is the lowest resource id.

Key ideas of conflict resolution in distributed systems.

1. **Priority among agents in conflict.** Some agents win and others lose. Fair winning: every agent that wants to win gets to win *eventually*.

2. **Tokens.** An agent that holds a token knows that other agents don't hold the same token.

3. **Dynamic Data Structures**: Priorities based on ordering of resources, or timestamps, or dynamic partial-ordering (acyclic graphs) of agents