Supporting Nested Locking in Multiprocessor Real-Time Systems

Bryan C. Ward
James H. Anderson

Dept. of Computer Science
UNC Chapel Hill

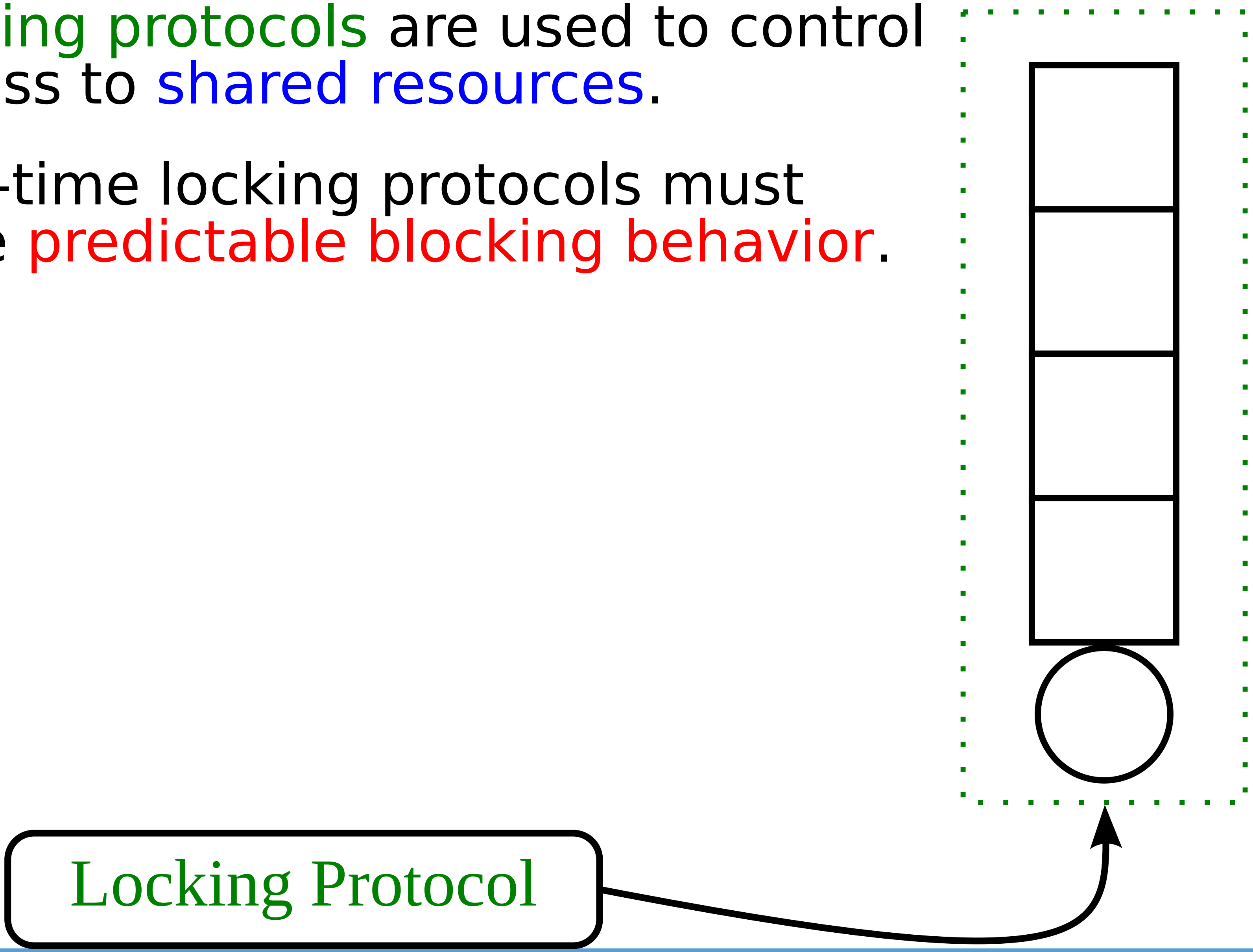
July 13, 2012

Real-Time Locking Protocols

- Locking protocols are used to control access to shared resources.
- Real-time locking protocols must have predictable blocking behavior.

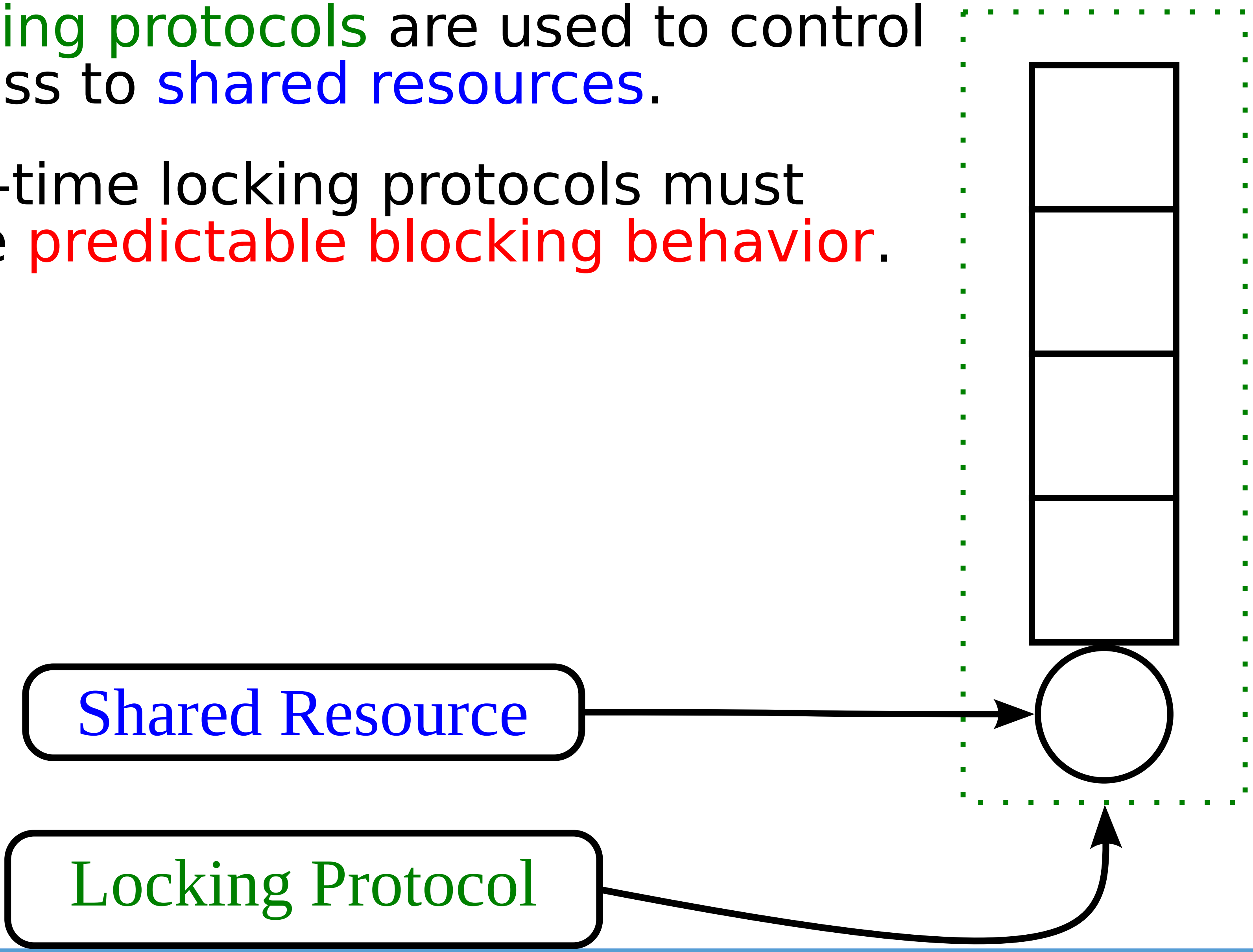
Real-Time Locking Protocols

- Locking protocols are used to control access to shared resources.
- Real-time locking protocols must have predictable blocking behavior.



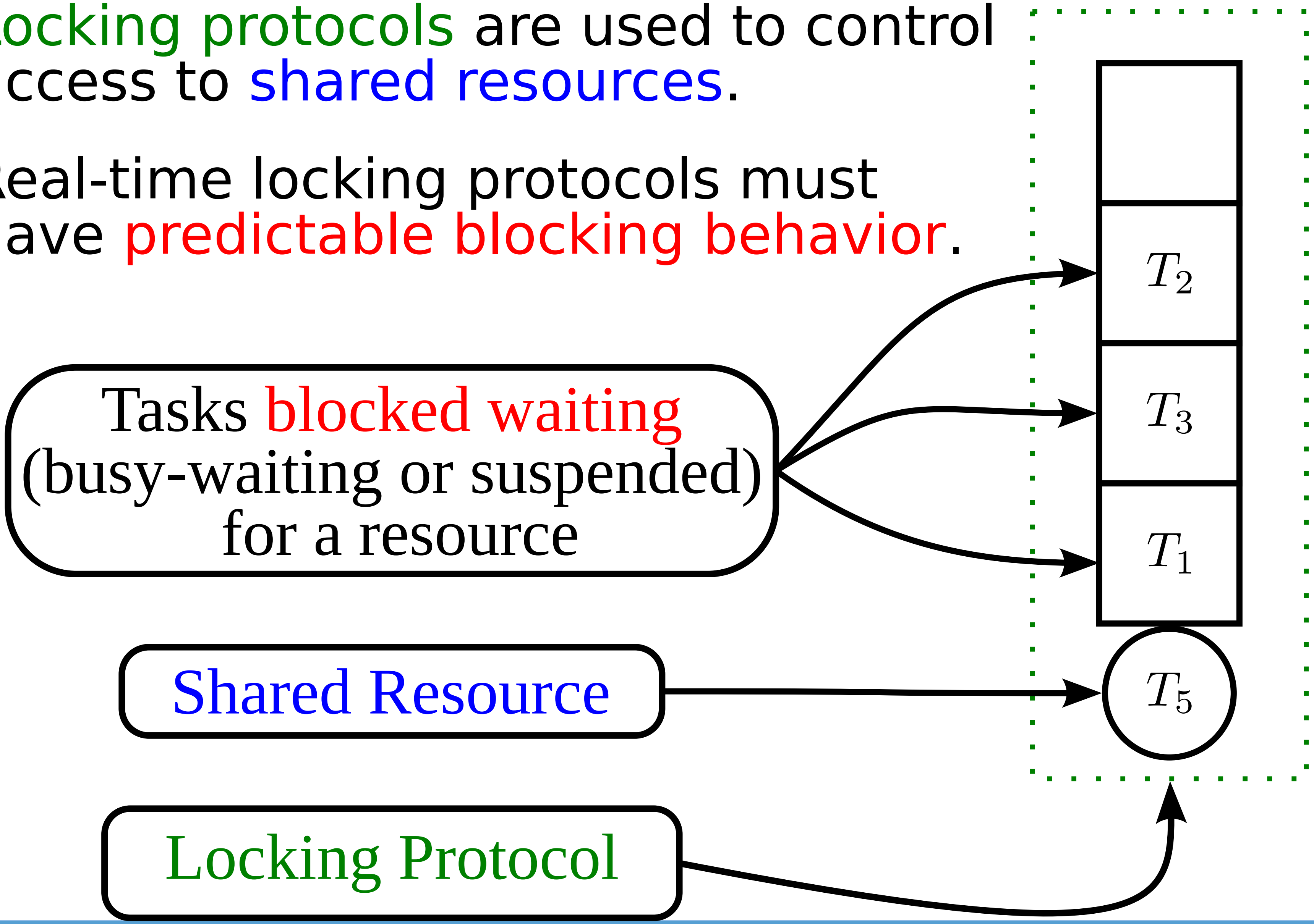
Real-Time Locking Protocols

- Locking protocols are used to control access to shared resources.
- Real-time locking protocols must have predictable blocking behavior.



Real-Time Locking Protocols

- Locking protocols are used to control access to shared resources.
- Real-time locking protocols must have predictable blocking behavior.



Nested Locks

- If a job holding a shared resource makes a resource request it is a nested request.
- Nested requests can allow resource holding jobs to be blocked.
- Nested requests can cause deadlock.
- No previous multiprocessor real-time locking protocols support nested resource requests.
  - Issue is avoided via group locks.
  - Group locks treat a set of resources as one.
  - Group locks can decrease parallelism.

Pi-Blocking

- A job experiences pi-blocking when it should be scheduled but is not.
- Three ways to measure pi-blocking:
  - Suspension-oblivious (s-oblivious).
  - Suspension-aware (s-aware).
  - Spin-based.

Pi-Blocking

- A job experiences pi-blocking when it should be scheduled but is not.
- Three ways to measure pi-blocking:
  - Suspension-oblivious (s-oblivious).
  - Suspension-aware (s-aware).
  - Spin-based.

Only applies to suspension-based locks

Pi-Blocking

- A job experiences pi-blocking when it should be scheduled but is not.
- Three ways to measure pi-blocking:
  - Suspension-oblivious (s-oblivious).
  - Suspension-aware (s-aware).
  - Spin-based.

Only applies to suspension-based locks

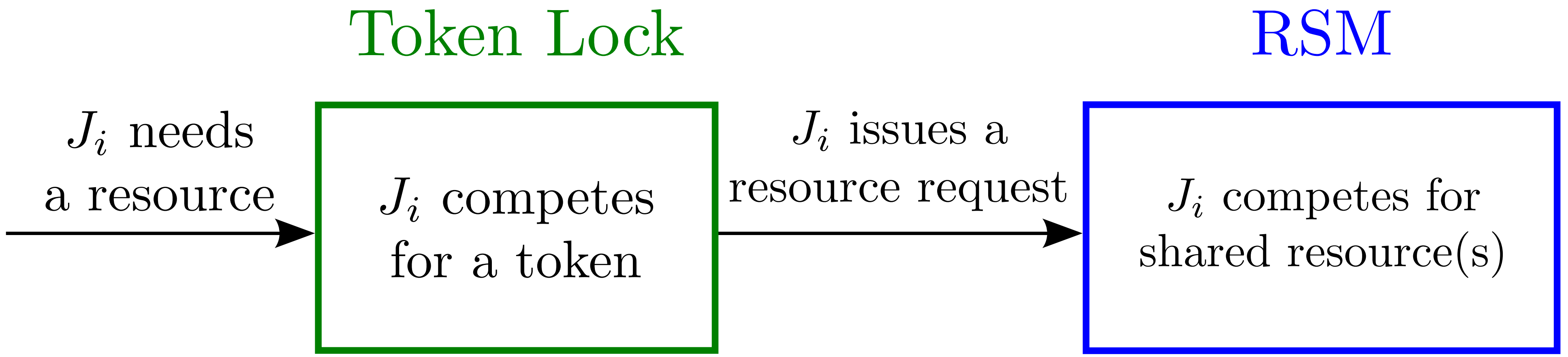
Only applies to spin-based locks

RNLP Architecture

- A job must acquire a token from a token lock before it can issue a resource request.
- A request satisfaction mechanism (RSM) orders the satisfaction of resource requests.
- Different token locks and RSMs can be paired on different platforms.

RNLP Architecture

- A job must acquire a token from a token lock before it can issue a resource request.
- A request satisfaction mechanism (RSM) orders the satisfaction of resource requests.
- Different token locks and RSMs can be paired on different platforms.



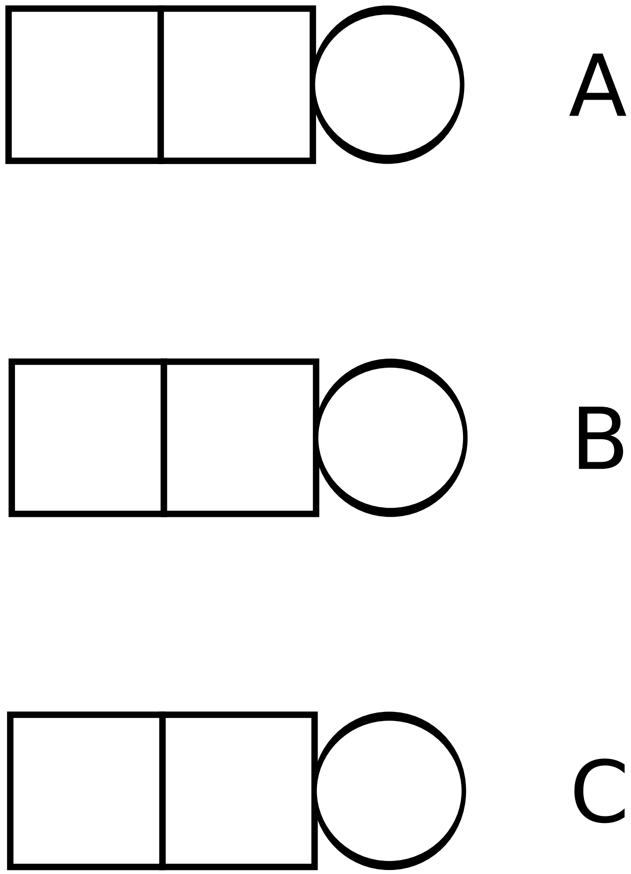
Token Locks

- Requirements of a token lock:
  - No more than *k* jobs can hold a token at a time.
  - A pi-blocked job makes progress.
- Can use existing *k*-exclusion locks.
  - O-KGLP (Elliott and Anderson, RTNS 2011)
  - Clustered k-exclusion OMLP (CK-OMLP) (Brandenburg and Anderson, EMSOFT 2011)
- We also developed the I-KGLP.

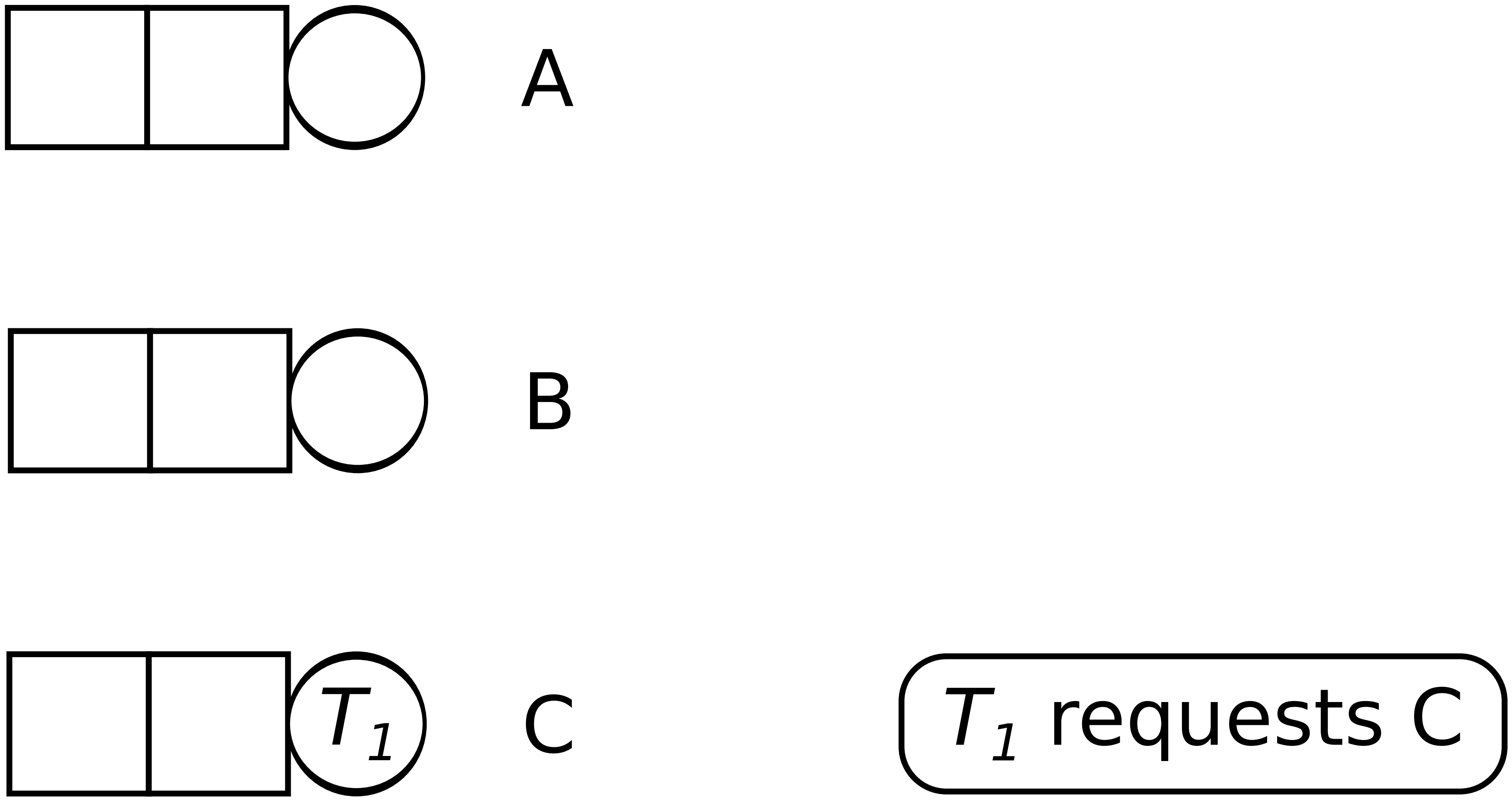
RSM

- Each resource *A* has a queue RQ_A.
- RQs are ordered by timestamp of token acquisition.
- A job at the head of RQ_A might acquire *A*. The RNLP is not greedy.
- A job must wait even if it is at the head of a RQ if it could possibly block another job with an earlier timestamp.

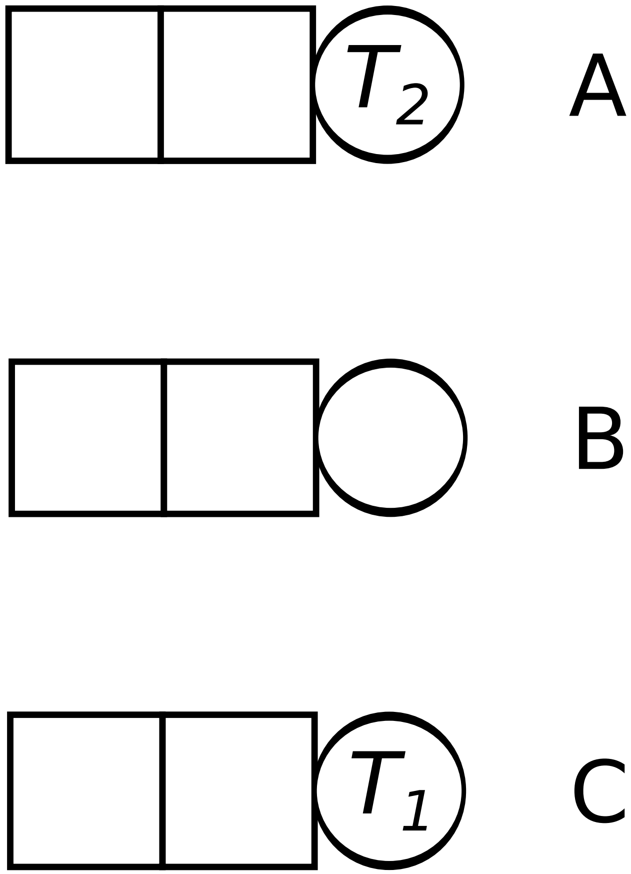
RSM Example



RSM Example

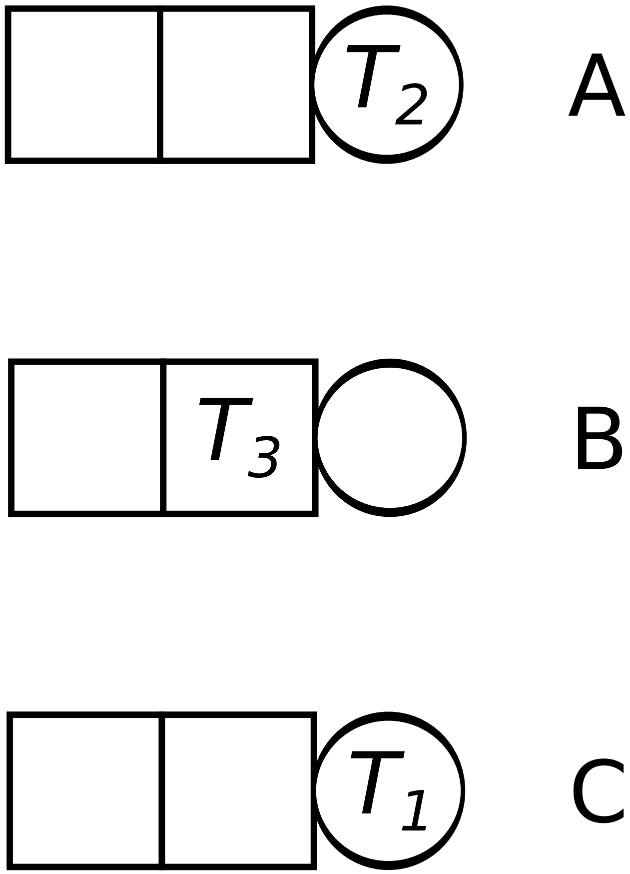


RSM Example



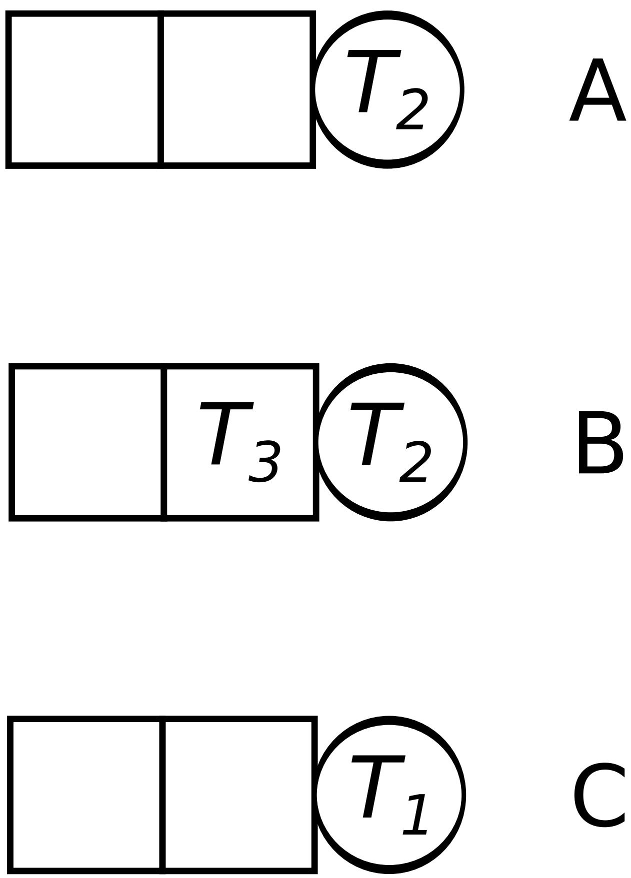
T_2 requests A

RSM Example



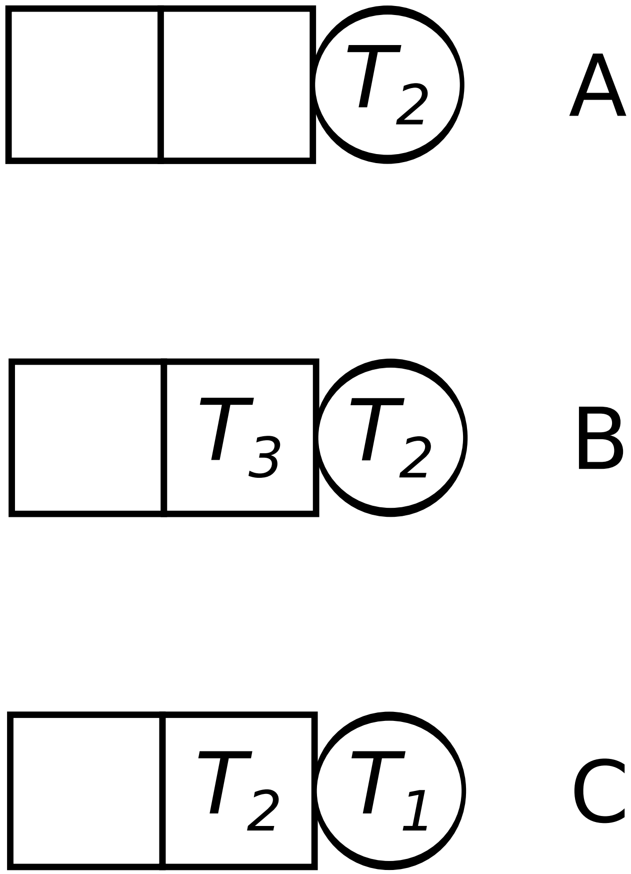
T_3 requests but does not acquire B because T_2 may request B in the future.

RSM Example



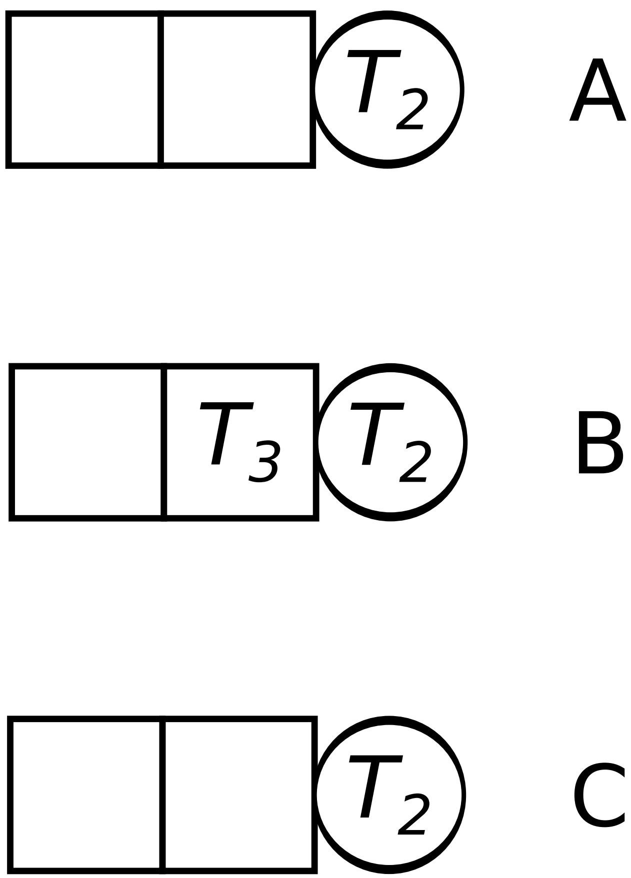
T_2 requests and acquires B because it has an earlier timestamp than T_3.

RSM Example



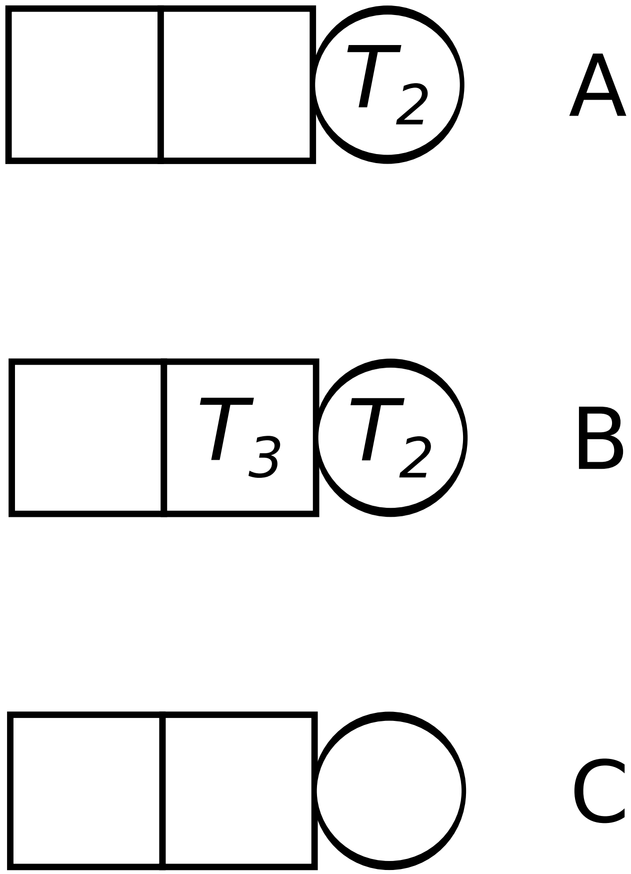
T_2 requests C but is blocked by T_1.

RSM Example



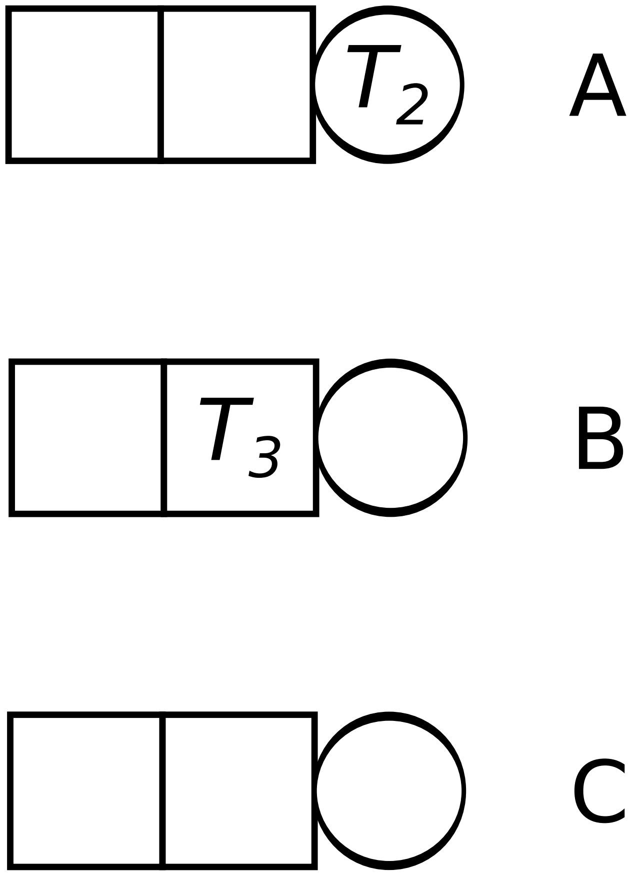
T_1 releases C and T_2 acquires it.

RSM Example



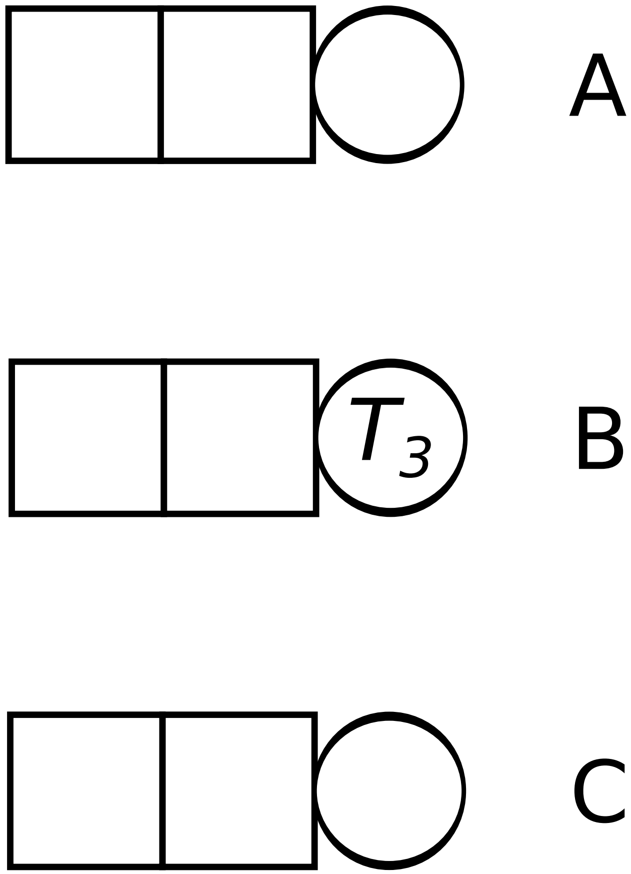
T_2 releases C

RSM Example



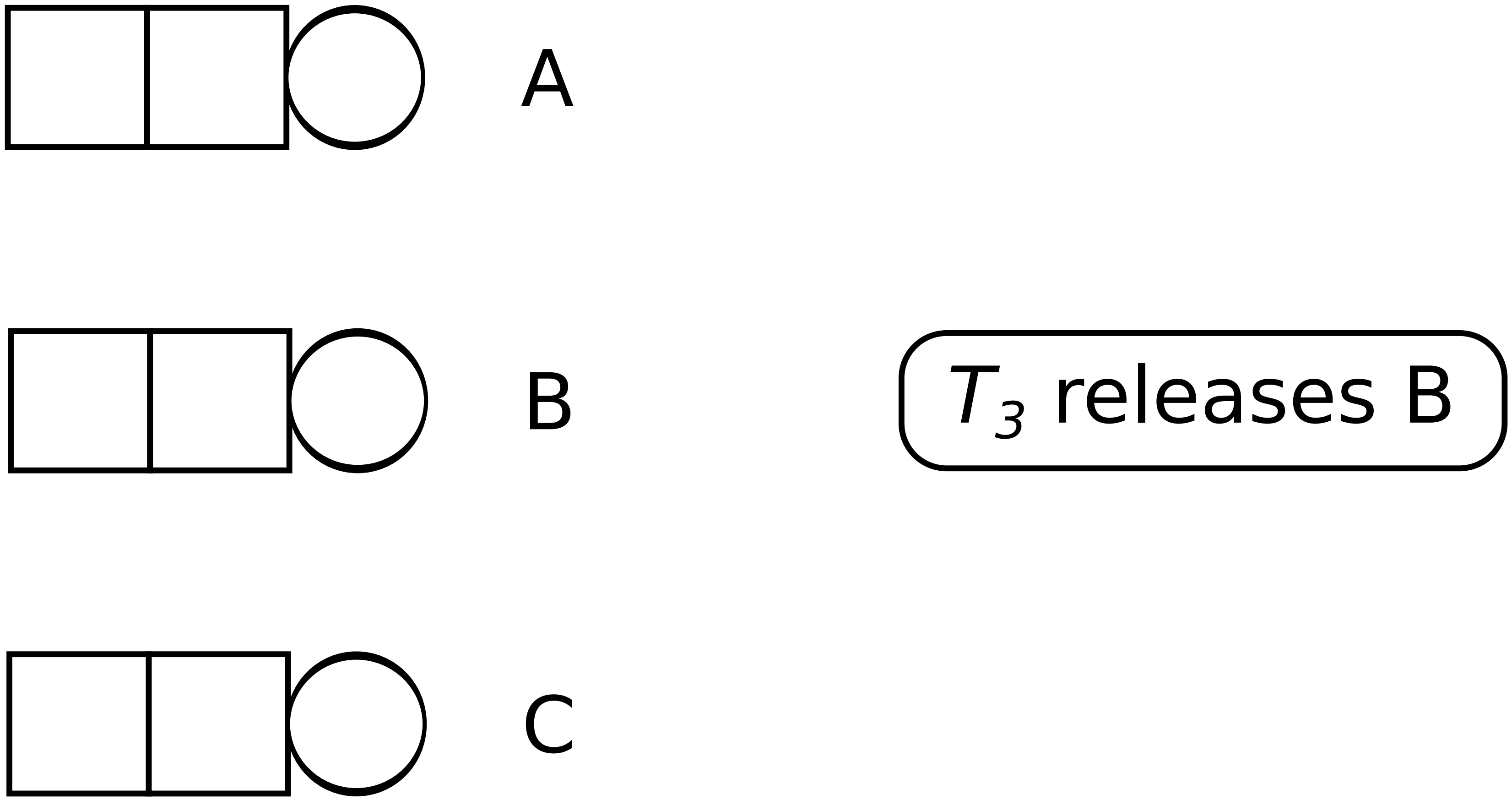
T_2 releases B but T_3 still can't acquire it.

RSM Example



T_2 releases A and T_3 can then acquire B.

RSM Example



Progress Mechanisms

- Progress mechanisms are used to ensure progress.
- RNLP compatible with three progress mechanisms:
  - Priority Inheritence: a resource holding job inherits another waiting job's priority.
  - Priority Boosting: a resource holding job's priority is boosted above all other jobs.
  - Priority Donation: a hybrid of boosting and inheritence.

Progress Mechanisms

- Progress mechanisms are used to ensure progress.
- RNLP compatible with three progress mechanisms:
  - Priority Inheritence: a resource holding job inherits another waiting job's priority.
  - Priority Boosting: a resource holding job's priority is boosted above all other jobs.
  - Priority Donation: a hybrid of boosting and inheritence.

Progress mechanisms can cause jobs not engaged in the locking protocol to be pi-blocked.

Boosting and Donation

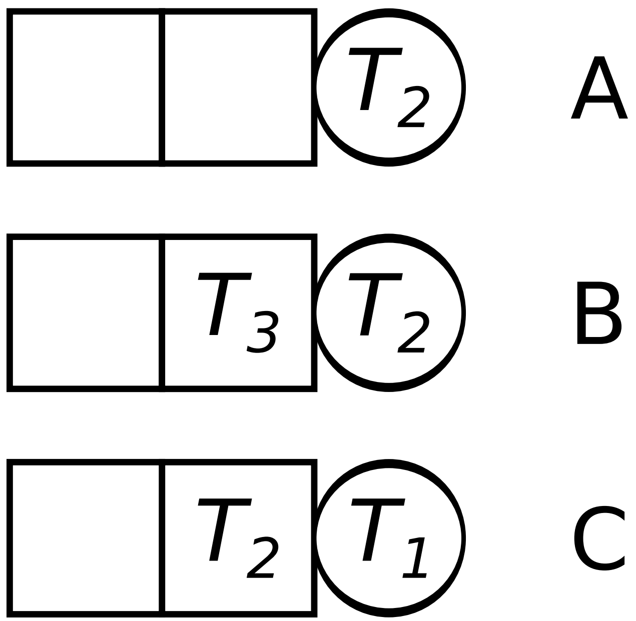
- Priority Boosting: the earliest *m* timestamp resource-holding jobs are priority boosted.
- Priority Donation:
  - Must use CK-OMLP as token lock.
  - Priority donation ensures that the token holding jobs have the highest effective priorities in the system.

Priority Inheritence

- A job's priority can only be inherited by one other job at a time.
- A job's priority may be inherited by the earliest timestamp job blocking it.

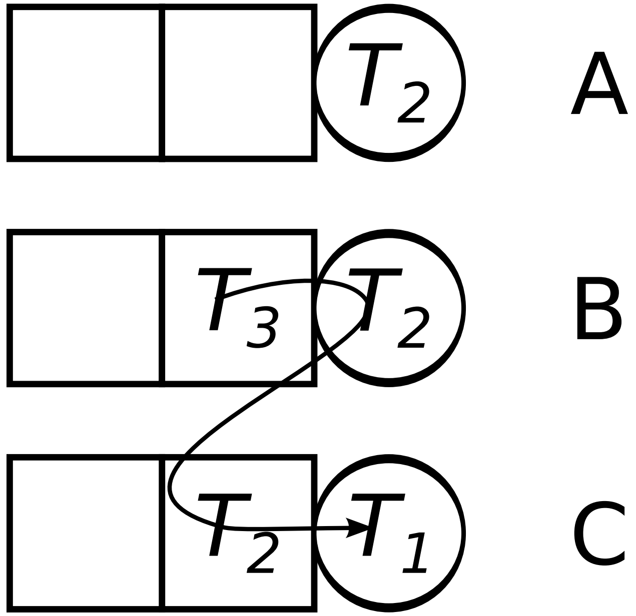
Priority Inheritence

- A job's priority can only be inherited by one other job at a time.
- A job's priority may be inherited by the earliest timestamp job blocking it.



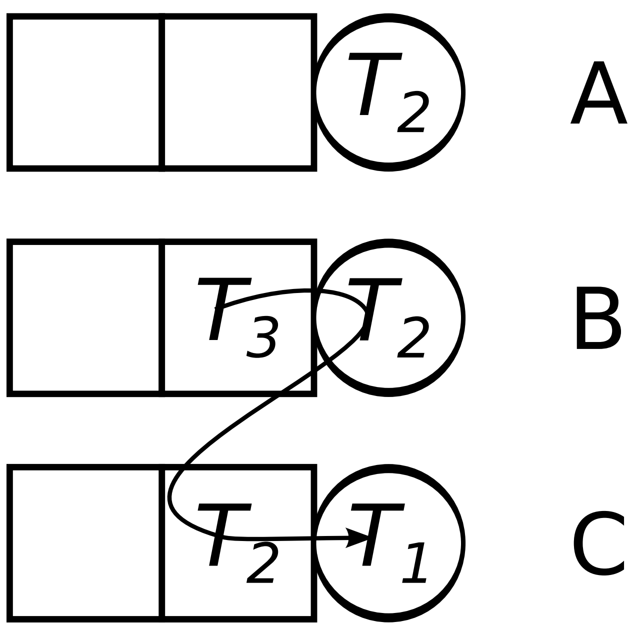
Priority Inheritence

- A job's priority can only be inherited by one other job at a time.
- A job's priority may be inherited by the earliest timestamp job blocking it.



Priority Inheritence

- A job's priority can only be inherited by one other job at a time.
- A job's priority may be inherited by the earliest timestamp job blocking it.



T_1 blocks both T_2 and T_3, and thus executes with the highest effective priority of T_1, T_2 and T_3.

Number of Tokens

- More tokens allow for the possibility of increased parallelism.
- Fewer tokens mean less pi-blocking in the RSM and more pi-blocking in the token lock.
- Number of tokens depends upon the analysis:
  - Spin-based: $k=m$.
  - S-aware: $k=n$.
  - S-oblivious: k=m.

Number of Tokens

- More tokens allow for the possibility of increased parallelism.
- Fewer tokens mean less pi-blocking in the RSM and more pi-blocking in the token lock.
- Number of tokens depends upon the analysis:
  - Spin-based: $k=m$.
  - S-aware: $k=n$.
  - S-oblivious: k=m.

A job acquires a token immediately upon request. We call this the trivial token lock (TTL).

Pairing

Analysis	Scheduler	Token Lock	k	RSM	Every Job Pi-blocking	Per-Request Pi-blocking
spin	Any	TTL	m	S-RSM	mL^{max}	$(m-1)L^{max}$
s-aware	Partitioned	TTL	n	B-RSM	nL^{max}	$(n-1)L^{max}$
	Clustered	TTL	n	B-RSM	$O(\phi \cdot n)$	$(n-1)L^{max}$
	Global[†]	TTL	n	I-RSM	$O(n)$	$(n-1)L^{max}$
s-oblivious	Partitioned	CK-OMLP	m	D-RSM	mL^{max}	$(m-1)L^{max}$
	Clustered	CK-OMLP	m	D-RSM	mL^{max}	$(m-1)L^{max}$
	Global	CK-OMLP	m	D-RSM	mL^{max}	$(m-1)L^{max}$
		O-KGLP	m	I-RSM	0	$(5m-1)L^{max}$
		I-KGLP	m	I-RSM	0	$(2m-1)L^{max}$

[†] Applicable only under certain schedulers.

Pairing

Analysis	Scheduler	Token Lock	k	RSM	Every Job Pi-blocking	Per-Request Pi-blocking
spin	Any	TTL	m	S-RSM	mL^{max}	$(m-1)L^{max}$
s-aware	Partitioned	TTL	n	B-RSM	nL^{max}	$(n-1)L^{max}$
	Clustered	TTL	n	B-RSM	$O(\phi \cdot n)$	$(n-1)L^{max}$
	Global†	TTL	n	I-RSM	$O(n)$	$(n-1)L^{max}$
s-oblivious	Partitioned	CK-OMLP	m	D-RSM	mL^{max}	$(m-1)L^{max}$
	Clustered	CK-OMLP	m	D-RSM	mL^{max}	$(m-1)L^{max}$
	Global	CK-OMLP	m	D-RSM	mL^{max}	$(m-1)L^{max}$
		O-KGLP	m	I-RSM	0	$(5m-1)L^{max}$
		I-KGLP	m	I-RSM	0	$(2m-1)L^{max}$

† Applicable only under certain schedulers.

Type of progress mechanism employed.

Pairing

Analysis	Scheduler	Token Lock	k	RSM	Every Job Pi-blocking	Per-Request Pi-blocking
spin	Any	TTL	m	S-RSM	mL^{max}	$(m-1)L^{max}$
s-aware	Partitioned	TTL	n	B-RSM	nL^{max}	$(n-1)L^{max}$
	Clustered	TTL	n	B-RSM	$O(\phi \cdot n)$	$(n-1)L^{max}$
	Global†	TTL	n	I-RSM	$O(n)$	$(n-1)L^{max}$
s-oblivious	Partitioned	CK-OMLP	m	D-RSM	mL^{max}	$(m-1)L^{max}$
	Clustered	CK-OMLP	m	D-RSM	mL^{max}	$(m-1)L^{max}$
	Global	CK-OMLP	m	D-RSM	mL^{max}	$(m-1)L^{max}$
		O-KGLP	m	I-RSM	0	$(5m-1)L^{max}$
		I-KGLP	m	I-RSM	0	$(2m-1)L^{max}$

† Applicable only under certain schedulers.

Duration of pi-blocking
any job may experience.

Pairing

Duration of pi-blocking a job may experience per outermost request.

Analysis	Scheduler	Token Lock	k	RSM	Every Job Pi-blocking	Per-Request Pi-blocking
spin	Any	TTL	m	S-RSM	mL^{max}	$(m-1)L^{max}$
s-aware	Partitioned	TTL	n	B-RSM	nL^{max}	$(n-1)L^{max}$
	Clustered	TTL	n	B-RSM	$O(\phi \cdot n)$	$(n-1)L^{max}$
	Global†	TTL	n	I-RSM	$O(n)$	$(n-1)L^{max}$
s-oblivious	Partitioned	CK-OMLP	m	D-RSM	mL^{max}	$(m-1)L^{max}$
	Clustered	CK-OMLP	m	D-RSM	mL^{max}	$(m-1)L^{max}$
	Global	CK-OMLP	m	D-RSM	mL^{max}	$(m-1)L^{max}$
		O-KGLP	m	I-RSM	0	$(5m-1)L^{max}$
		I-KGLP	m	I-RSM	0	$(2m-1)L^{max}$

† Applicable only under certain schedulers.

Pairing

Asymptotically Optimal

Analysis	Scheduler	Token Lock	k	RSM	Every Job Pi-blocking	Per-Request Pi-blocking
spin	Any	TTL	m	S-RSM	mL^{max}	$(m-1)L^{max}$
s-aware	Partitioned	TTL	n	B-RSM	nL^{max}	$(n-1)L^{max}$
	Clustered	TTL	n	B-RSM	$O(\phi \cdot n)$	$(n-1)L^{max}$
	Global†	TTL	n	I-RSM	$O(n)$	$(n-1)L^{max}$
s-oblivious	Partitioned	CK-OMLP	m	D-RSM	mL^{max}	$(m-1)L^{max}$
	Clustered	CK-OMLP	m	D-RSM	mL^{max}	$(m-1)L^{max}$
	Global	CK-OMLP	m	D-RSM	mL^{max}	$(m-1)L^{max}$
		O-KGLP	m	I-RSM	0	$(5m-1)L^{max}$
		I-KGLP	m	I-RSM	0	$(2m-1)L^{max}$

† Applicable only under certain schedulers.

Conclusions

- The RNLP is the first multiprocessor real-time locking protocol supporting nested resource requests.

- The RNLP has maximum pi-blocking no worse than existing single-resource locking protocols.
  - The RNLP is optimal under all systems and types of analysis for which an optimal locking protocol is known.
  - Future progress mechanisms or k-exclusion locks can be incorporated to improve the RNLP.

Ongoing Work

- Support nested reader-writer and multi-unit resources.
- Develop a progress mechanism for clustered systems that yields an optimal RNLP variant under s-aware analysis.
- More detailed analysis to reflect the benefit of increased parallelism.
- Experimental evaluations.

Thank You!