Local Linearizability

Ana Sokolova UNIVERSITY of SALZBURG

joint work with:

Andreas Haas Google
Andreas Holzer UNIVERSITY OF TORONTO
Michael Lippautz Google
Ali Sezgin UNIVERSITY OF CAMBRIDGE
Tom Henzinger IST AUSTRIA
Christoph Kirsch UNIVERSITY of SALZBURG
Hannes Payer Google
Helmut Veith TU WIEN

Rigorous methods
for
engineering of
and
reasoning about
reactive systems

Rigorous methods
for
engineering of
and
reasoning about
reactive systems

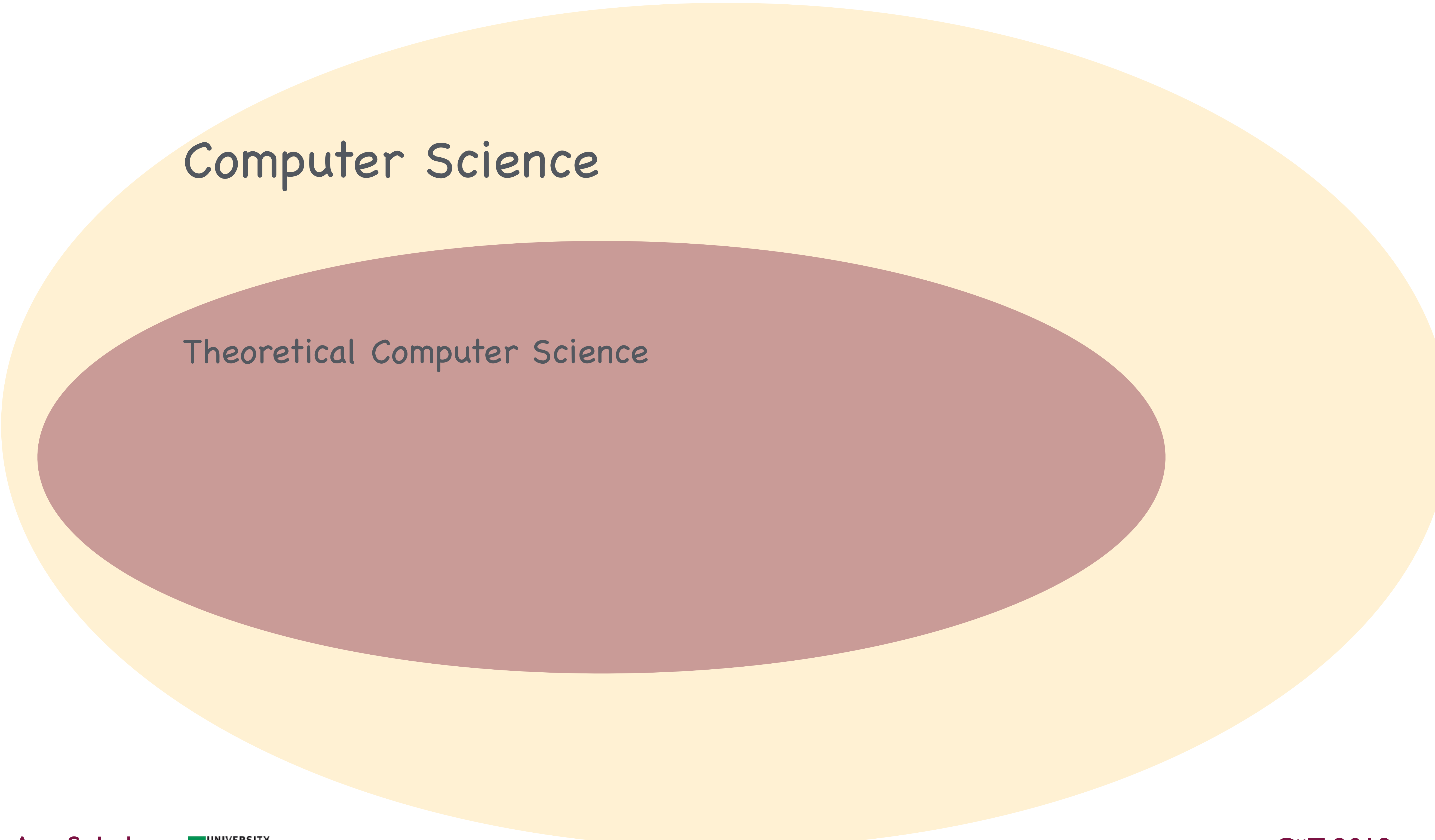
concurrent

Background big picture

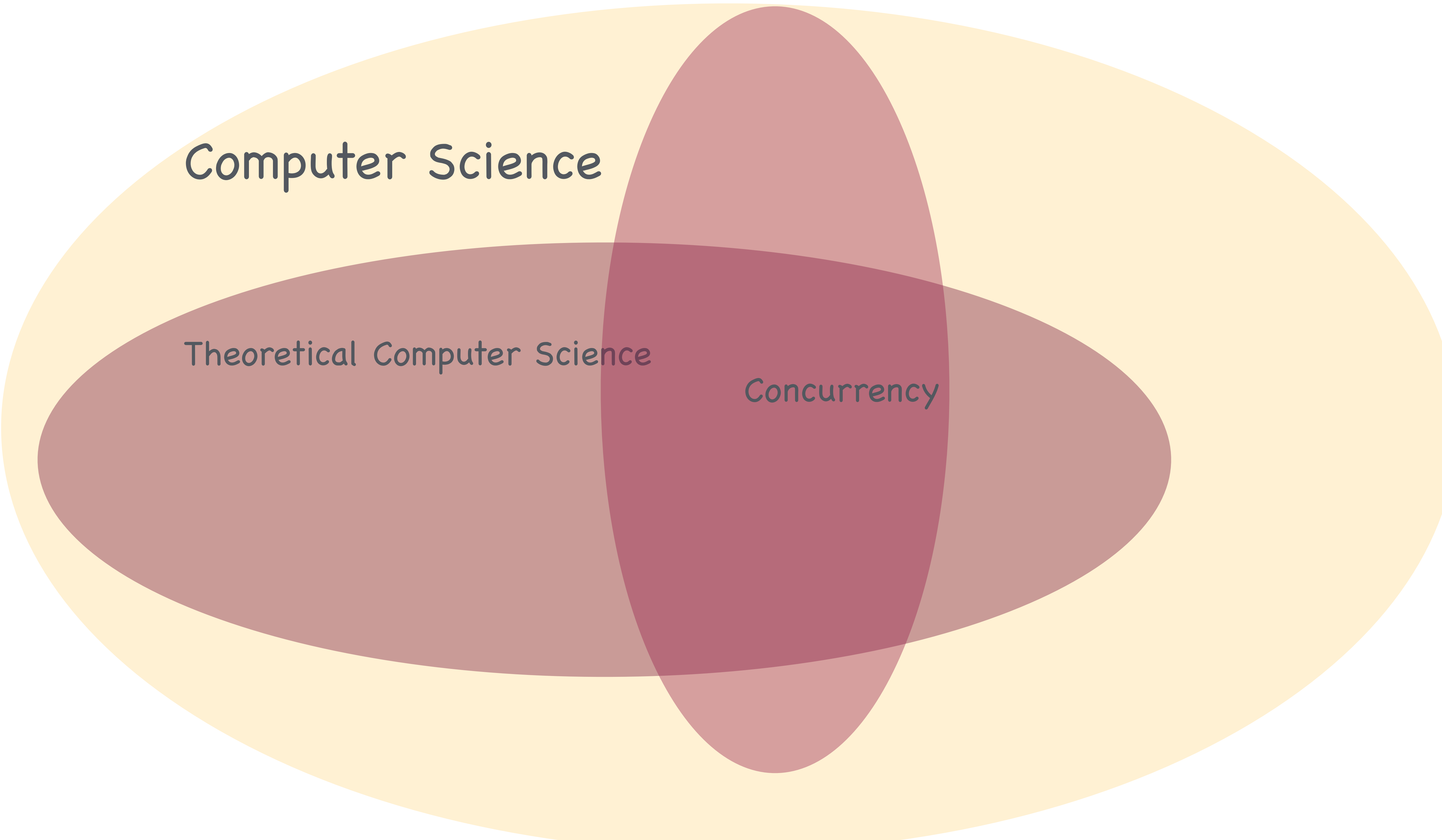
Background big picture

Computer Science

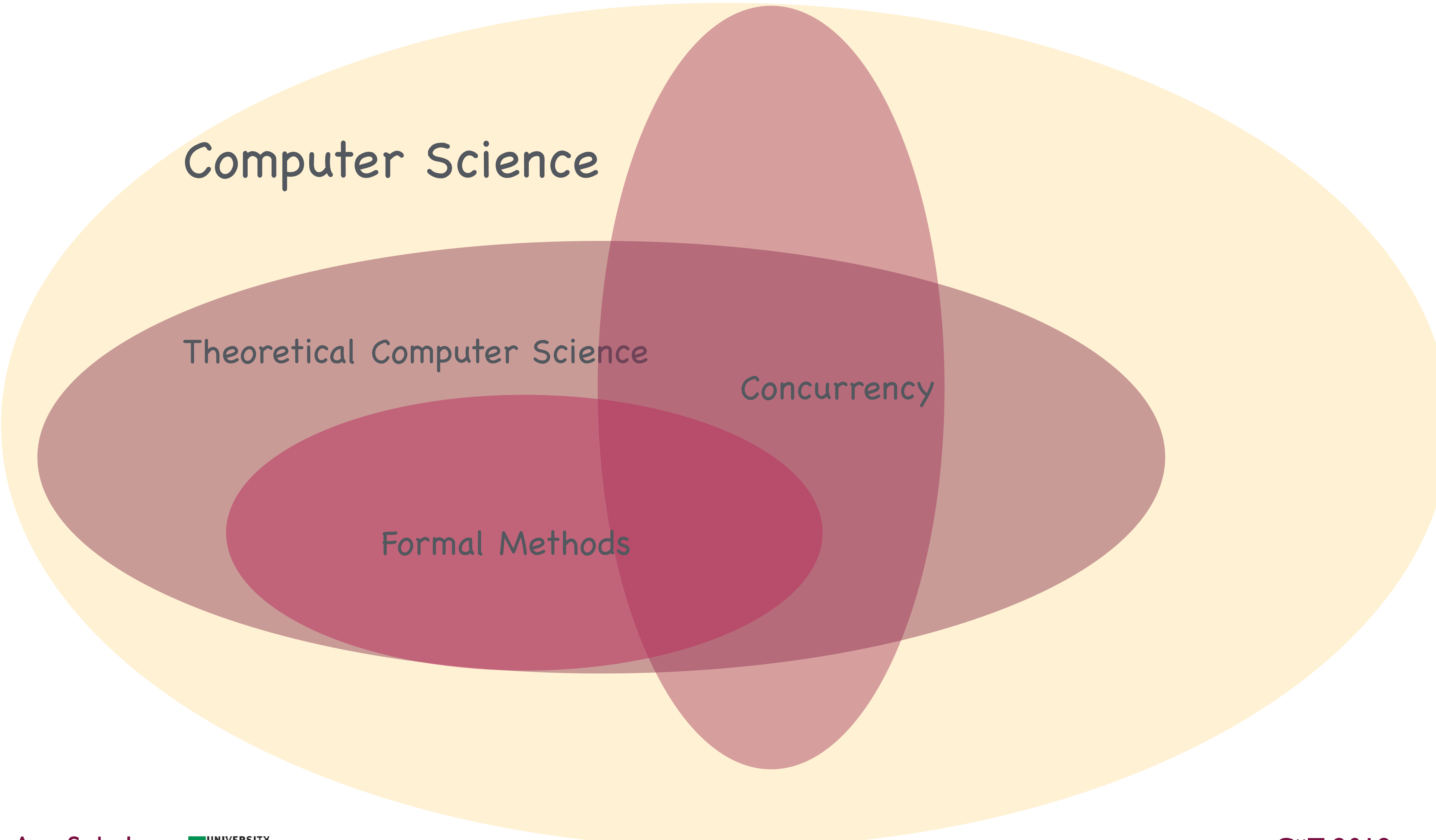
Background big picture



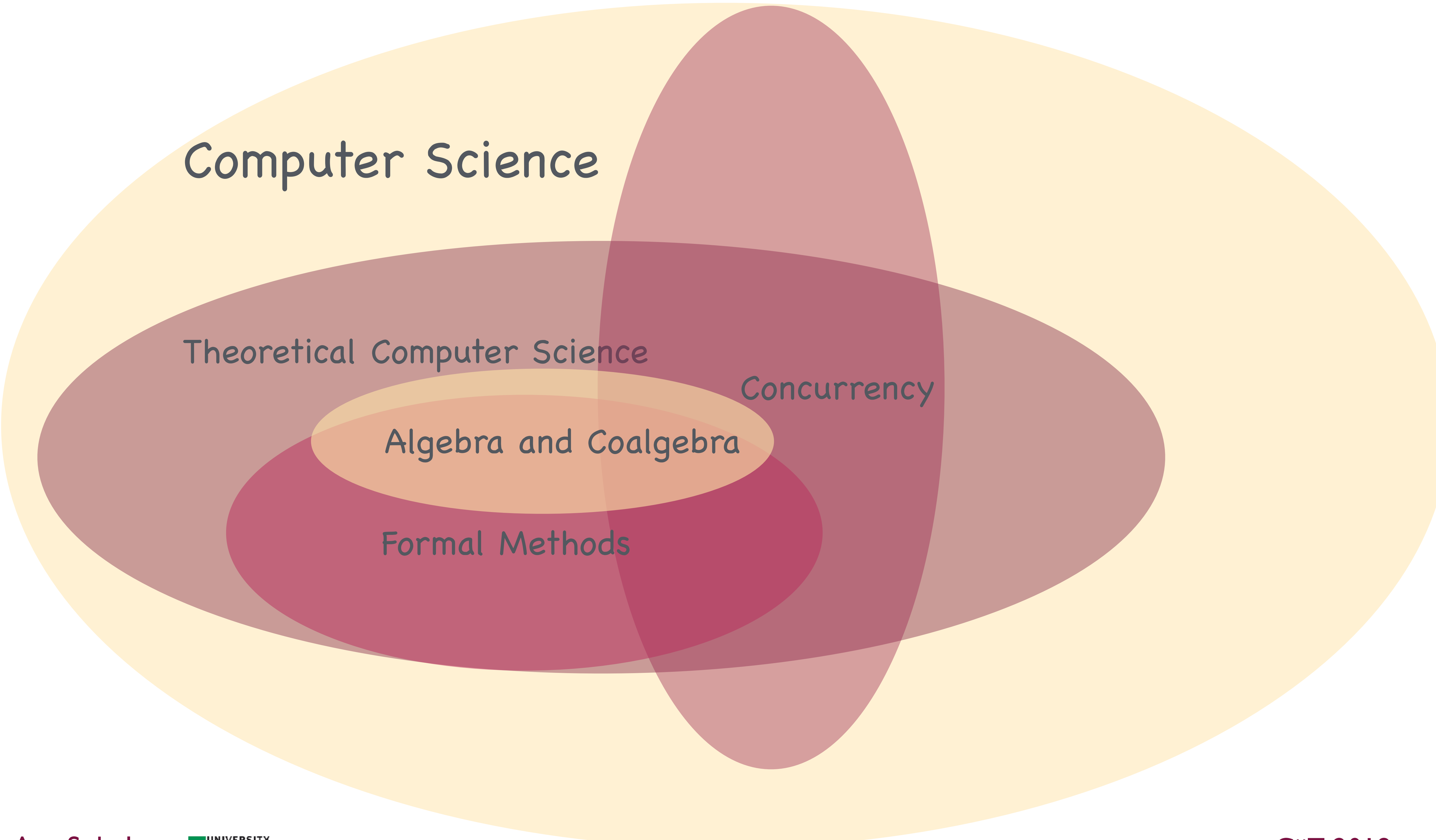
Background big picture



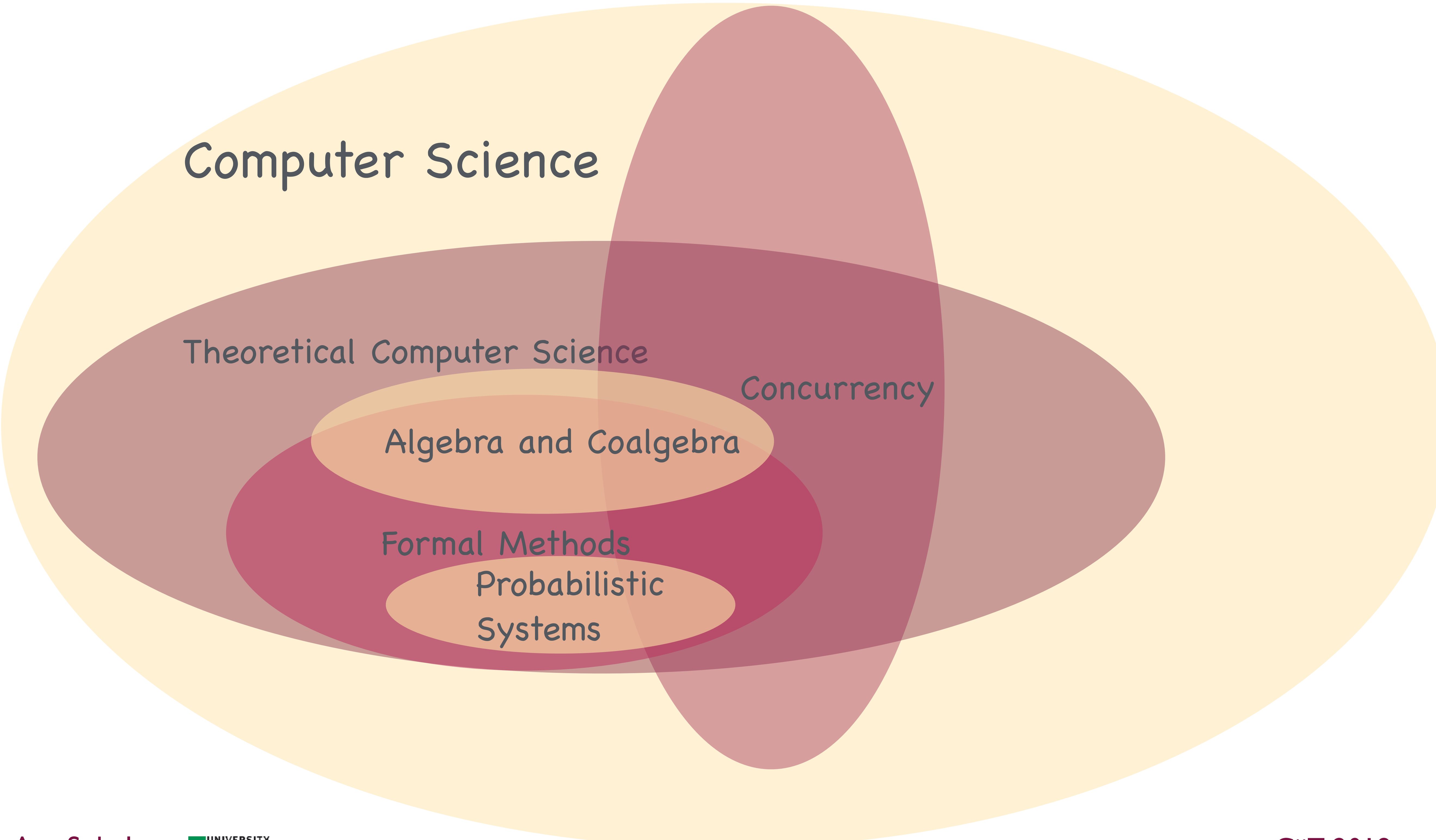
Background big picture



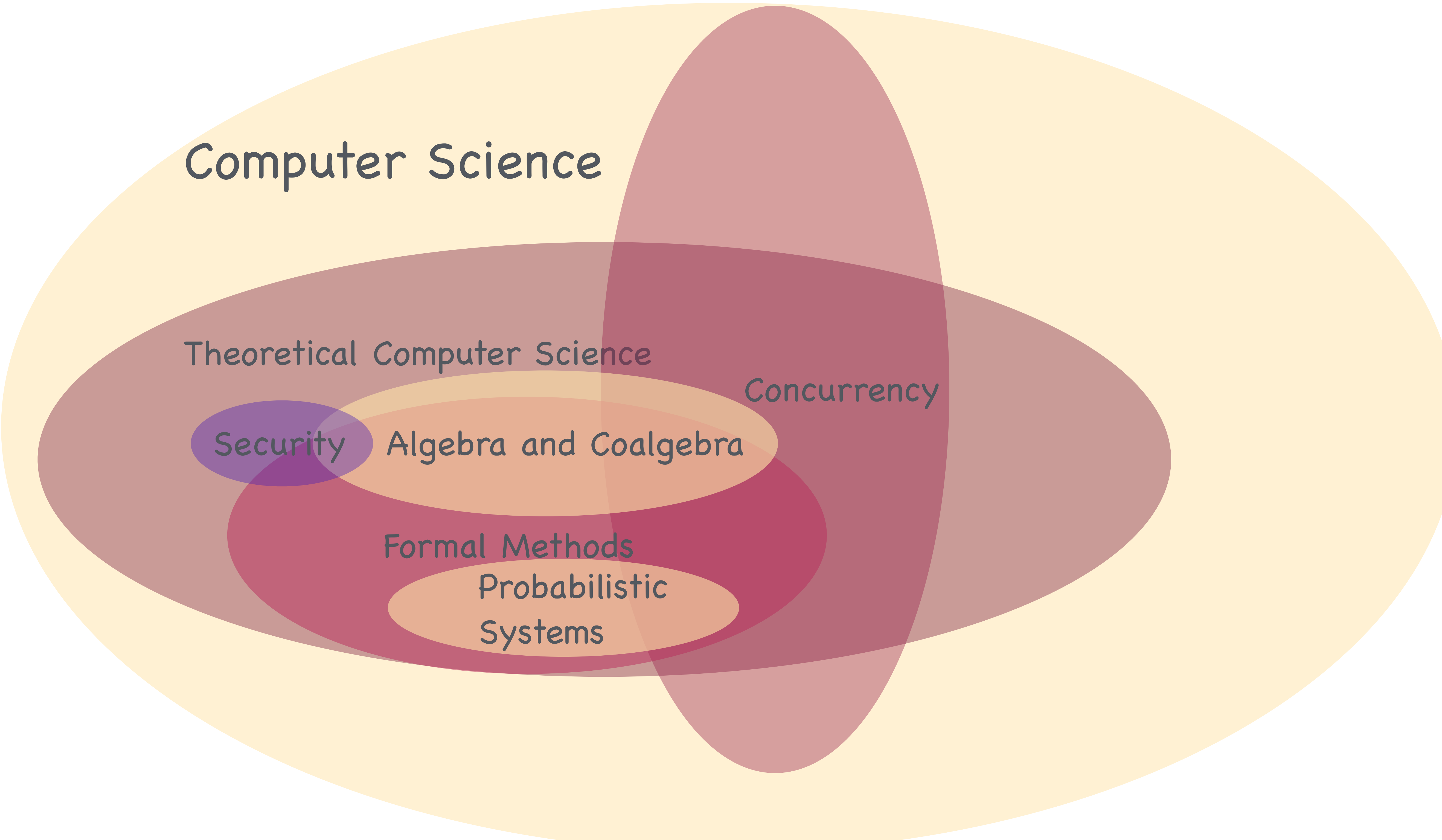
Background big picture



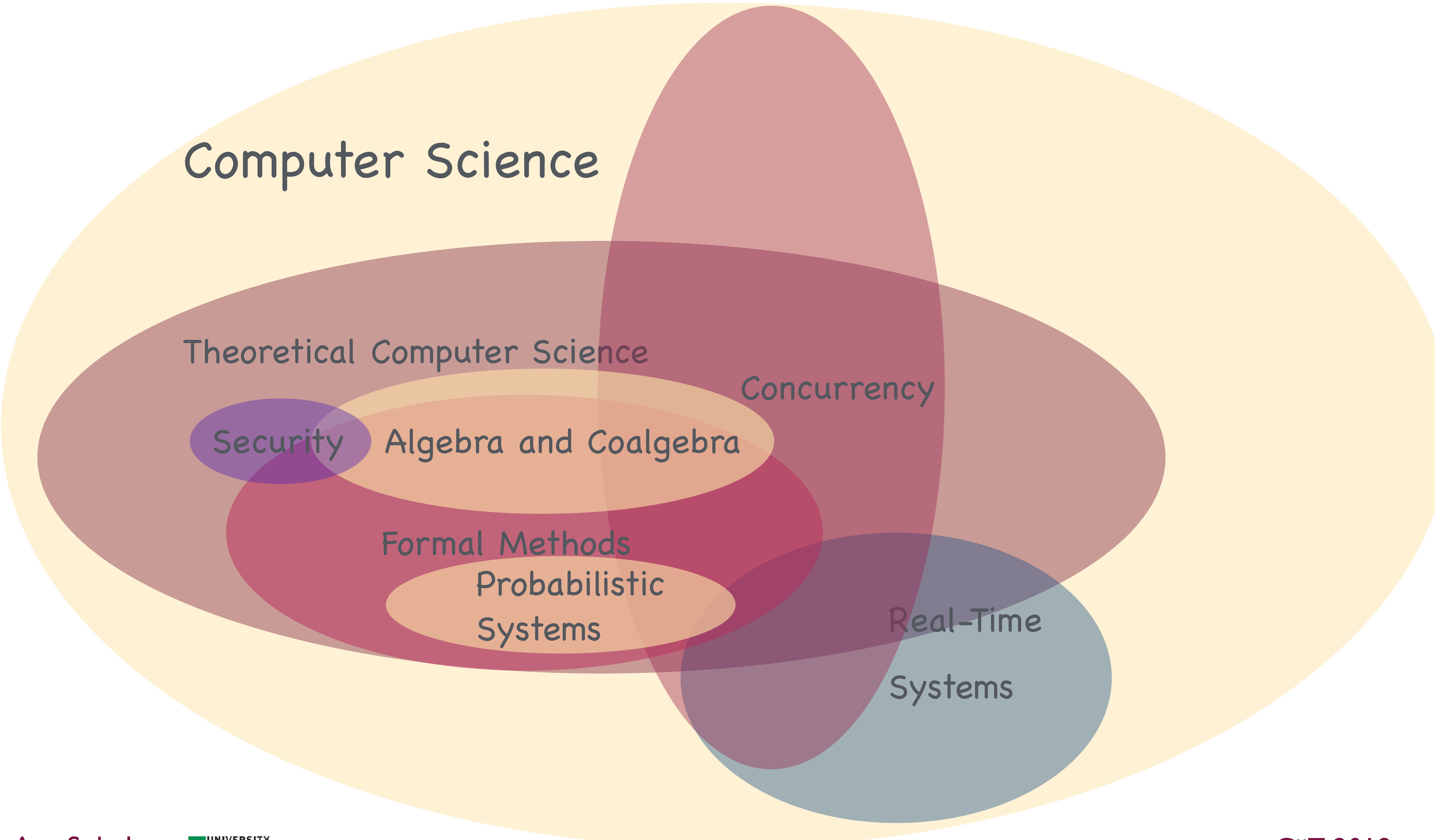
Background big picture



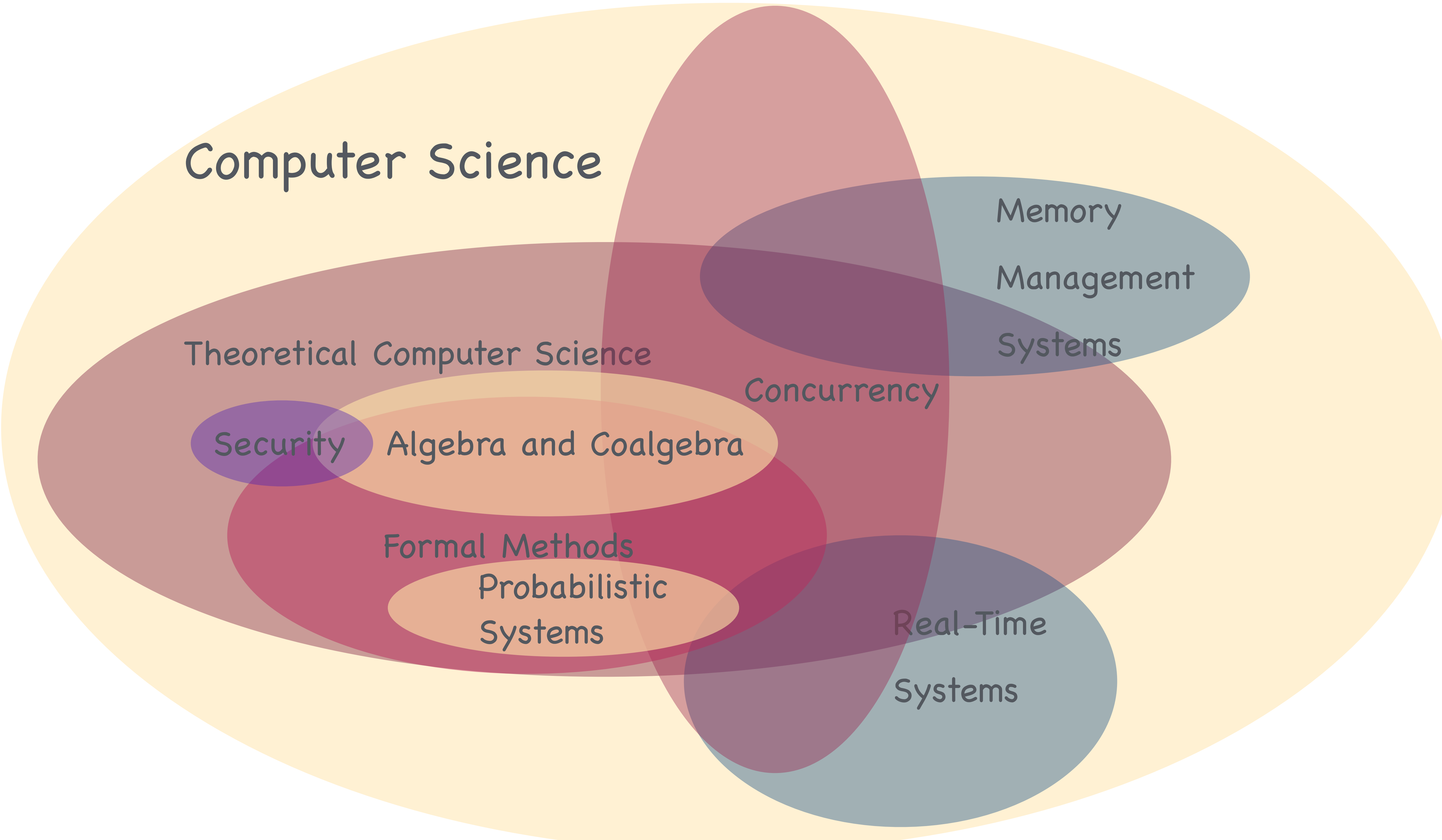
Background big picture



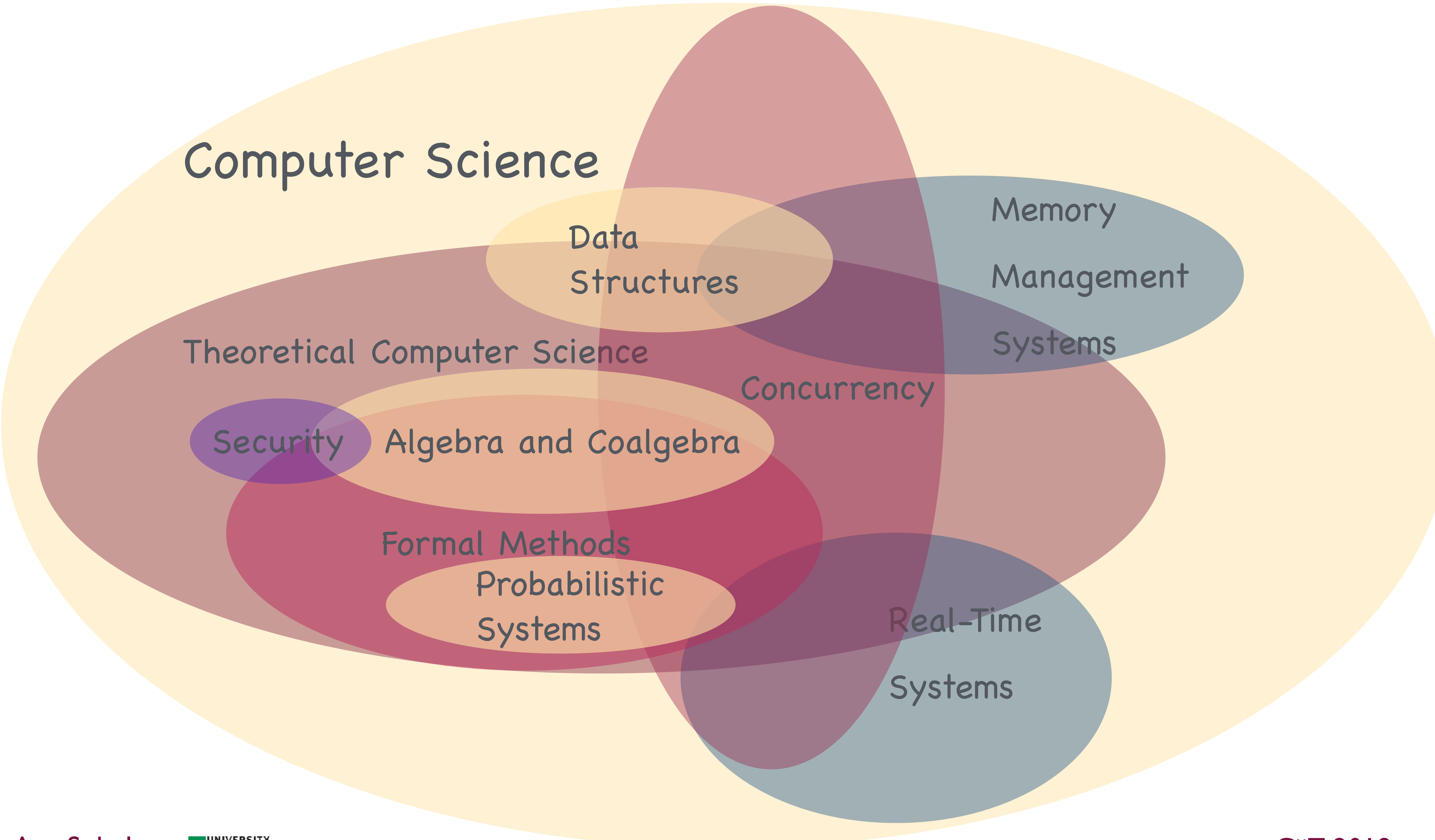
Background big picture



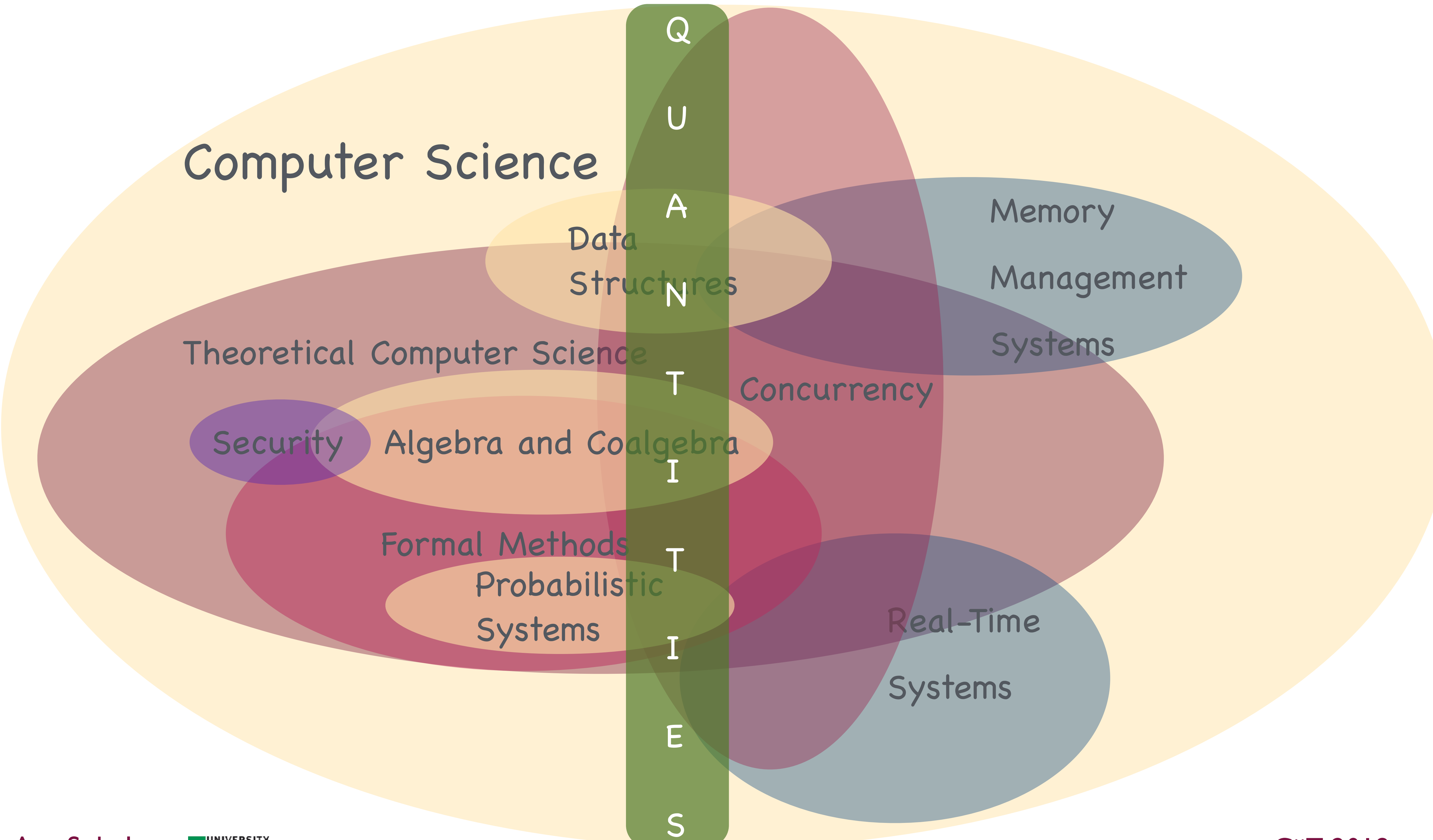
Background big picture



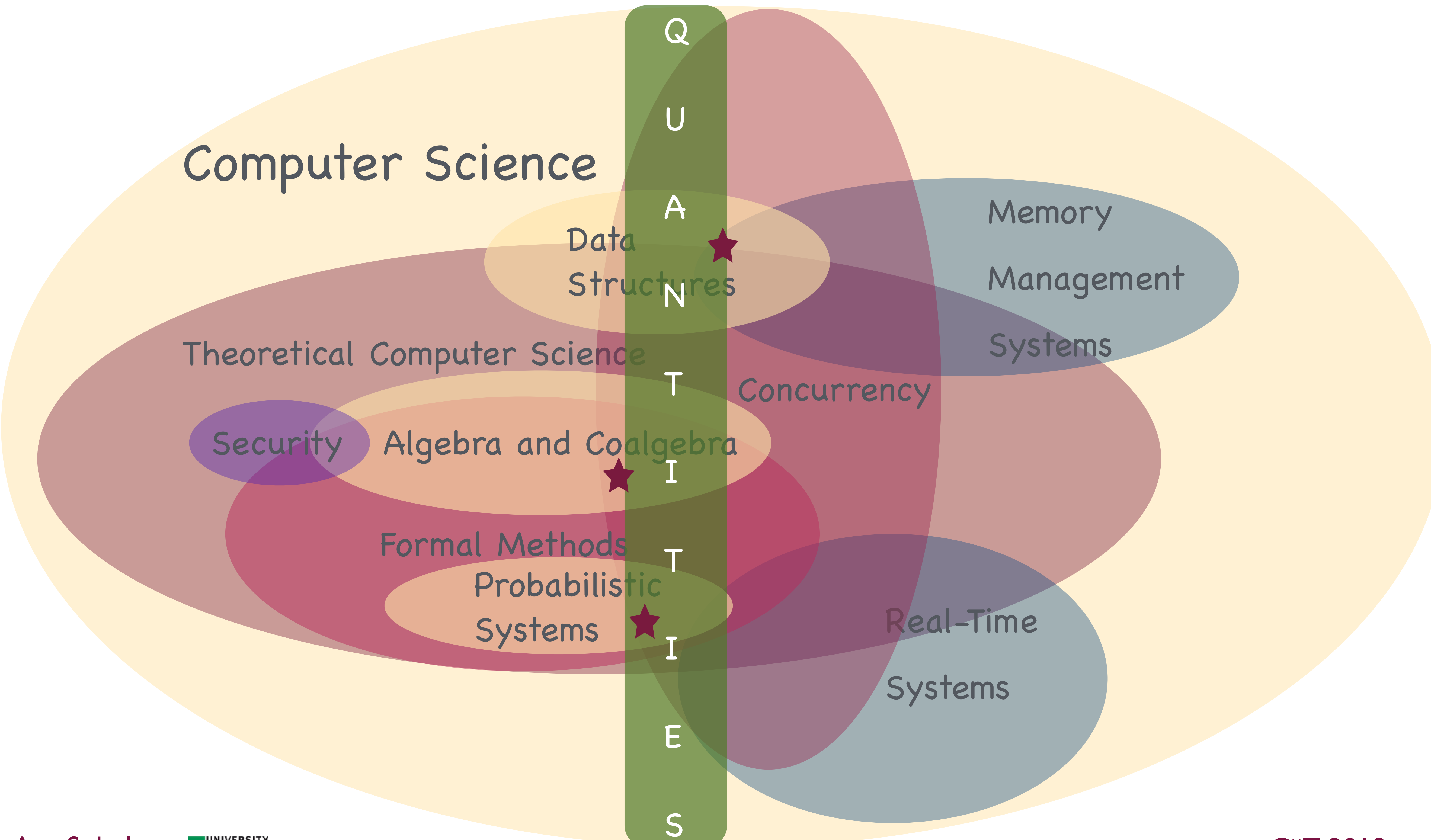
Background big picture



Background big picture



Current favourites



Local Linearizability

Ana Sokolova UNIVERSITY of SALZBURG

joint work with:

Andreas Haas Google
Andreas Holzer UNIVERSITY OF TORONTO
Michael Lippautz Google
Ali Sezgin UNIVERSITY OF CAMBRIDGE
Tom Henzinger IST AUSTRIA
Christoph Kirsch UNIVERSITY of SALZBURG
Hannes Payer Google
Helmut Veith TU WIEN

Concurrent Data Structures: Semantics and Relaxations

Concurrent Data Structures
Correctness and Performance

Concurrent Data Structures
Correctness and Performance

structure and power

Semantics of concurrent data structures

Semantics of concurrent data structures

e.g. pools, queues, stacks

Semantics of concurrent data structures

t1: enq(2) deq(1)

t2: enq(1) deq(2)

e.g. pools, queues, stacks

Semantics of concurrent data structures

t1: enq(2) deq(1)
t2: enq(1) deq(2)

e.g. pools, queues, stacks

- Sequential specification = set of legal sequences

- Consistency condition = e.g. linearizability / sequential consistency

Semantics of concurrent data structures

t1: enq(2) deq(1)

t2: enq(1) deq(2)

e.g. pools, queues, stacks

- Sequential specification = set of legal sequences

e.g. queue legal sequence
enq(1)enq(2)deq(1)deq(2)

- Consistency condition = e.g. linearizability / sequential consistency

Semantics of concurrent data structures

t1: enq(2) deq(1)

t2: enq(1) deq(2)

e.g. pools, queues, stacks

- Sequential specification = set of legal sequences

e.g. queue legal sequence
enq(1)enq(2)deq(1)deq(2)

- Consistency condition = e.g. linearizability / sequential consistency

e.g. the concurrent history above is a linearizable queue concurrent history

Consistency conditions

Linearizability [Herlihy,Wing '90]

↓

Sequential Consistency [Lamport'79]

Consistency conditions

there exists a legal sequence that preserves precedence

Linearizability [Herlihy,Wing ’90]

↓

Sequential Consistency [Lamport’79]

Consistency conditions

there exists a legal sequence that preserves precedence

Linearizability [Herlihy,Wing '90]

t1: enq(2) deq(1)

t2: enq(1) deq(2)

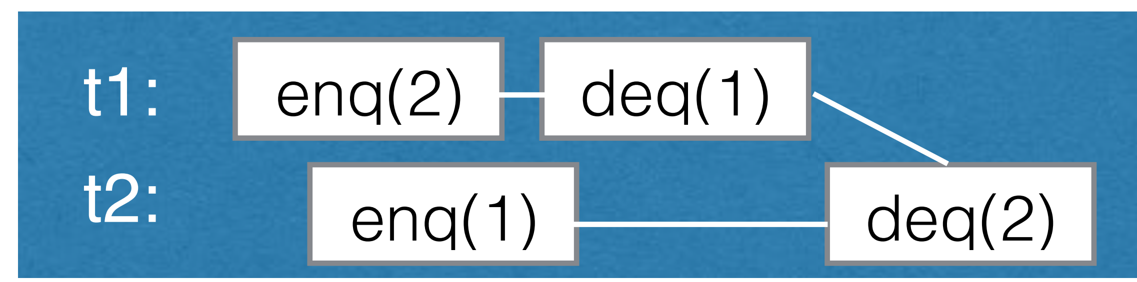
↓

Sequential Consistency [Lamport'79]

Consistency conditions

there exists a legal sequence that preserves precedence

Linearizability [Herlihy,Wing '90]

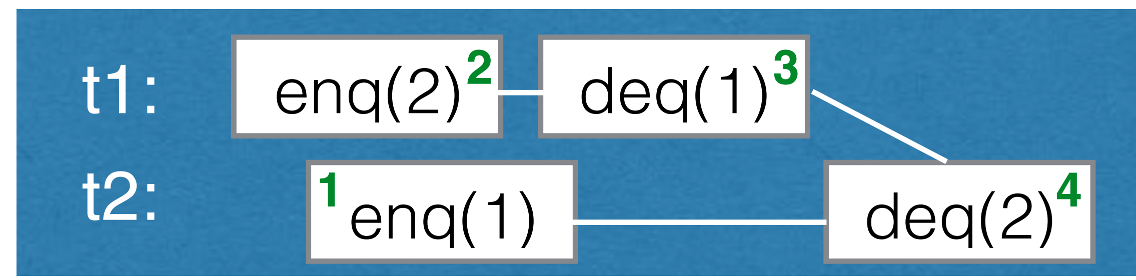


Sequential Consistency [Lamport'79]

Consistency conditions

there exists a legal sequence that preserves precedence

Linearizability [Herlihy,Wing '90]

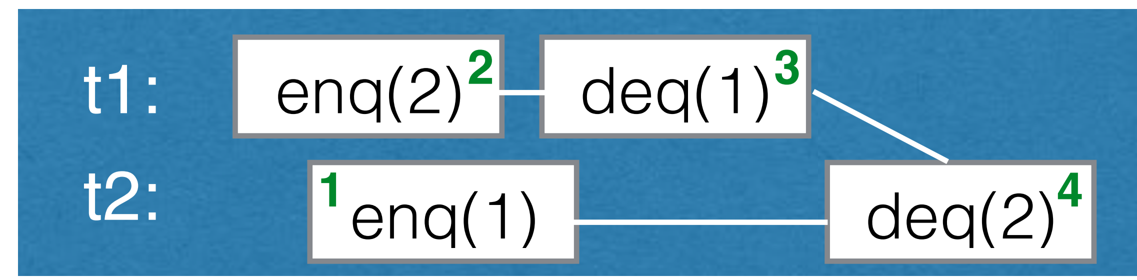


Sequential Consistency [Lamport'79]

Consistency conditions

there exists a legal sequence that preserves precedence

Linearizability [Herlihy,Wing '90]



↓

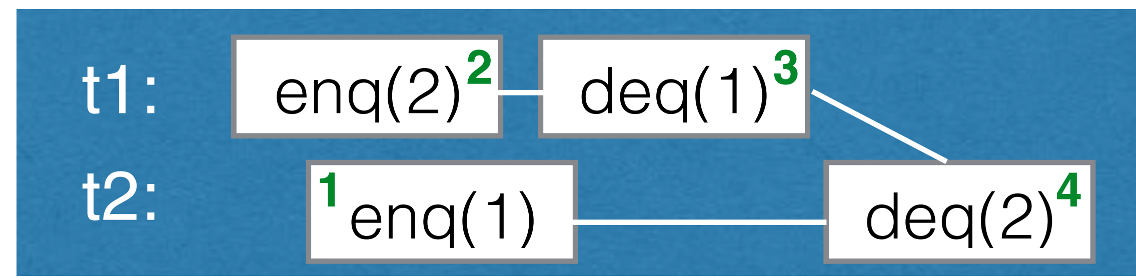
Sequential Consistency [Lamport'79]

there exists a legal sequence that preserves per-thread precedence (program order)

Consistency conditions

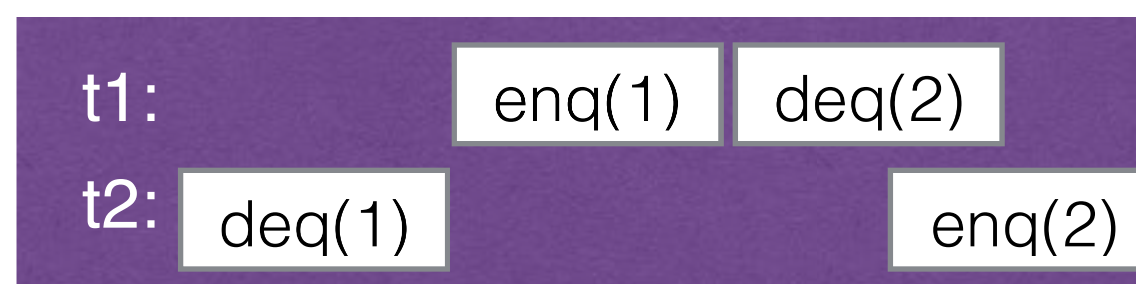
there exists a legal sequence that preserves precedence

Linearizability [Herlihy,Wing '90]



Sequential Consistency [Lamport'79]

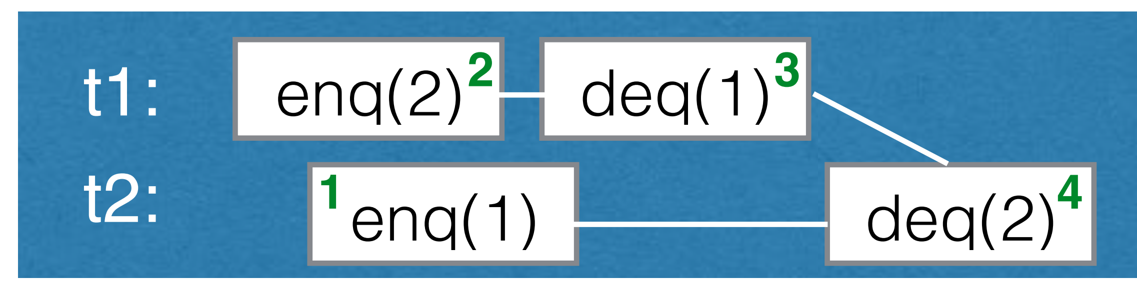
there exists a legal sequence that preserves per-thread precedence (program order)



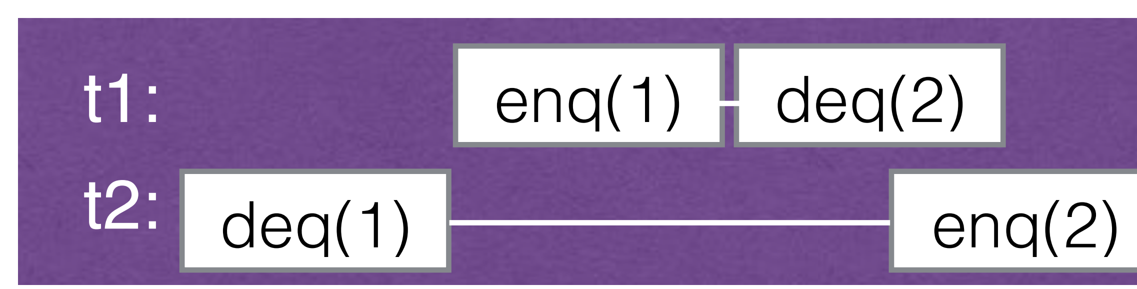
Consistency conditions

there exists a legal sequence that preserves precedence

Linearizability [Herlihy,Wing '90]



Sequential Consistency [Lamport'79]

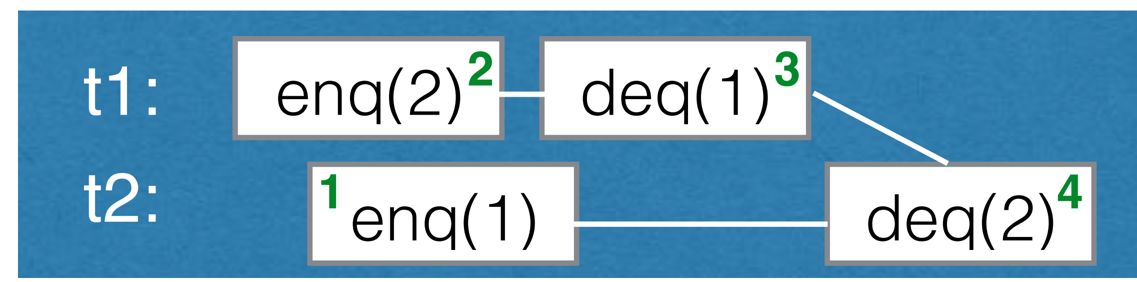


there exists a legal sequence that preserves per-thread precedence (program order)

Consistency conditions

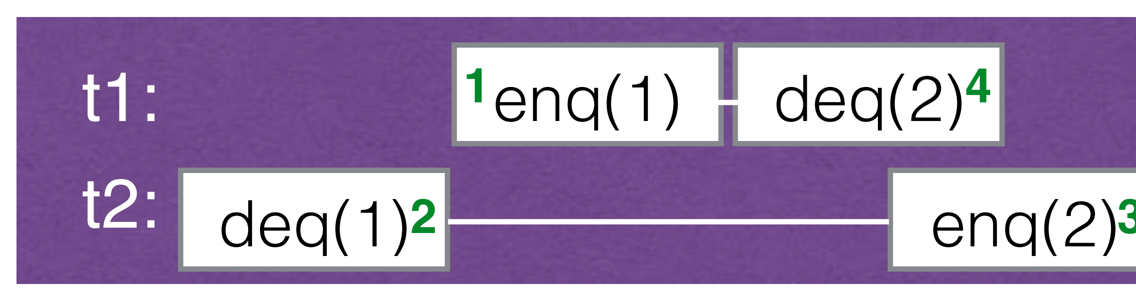
there exists a legal sequence that preserves precedence

Linearizability [Herlihy,Wing ’90]

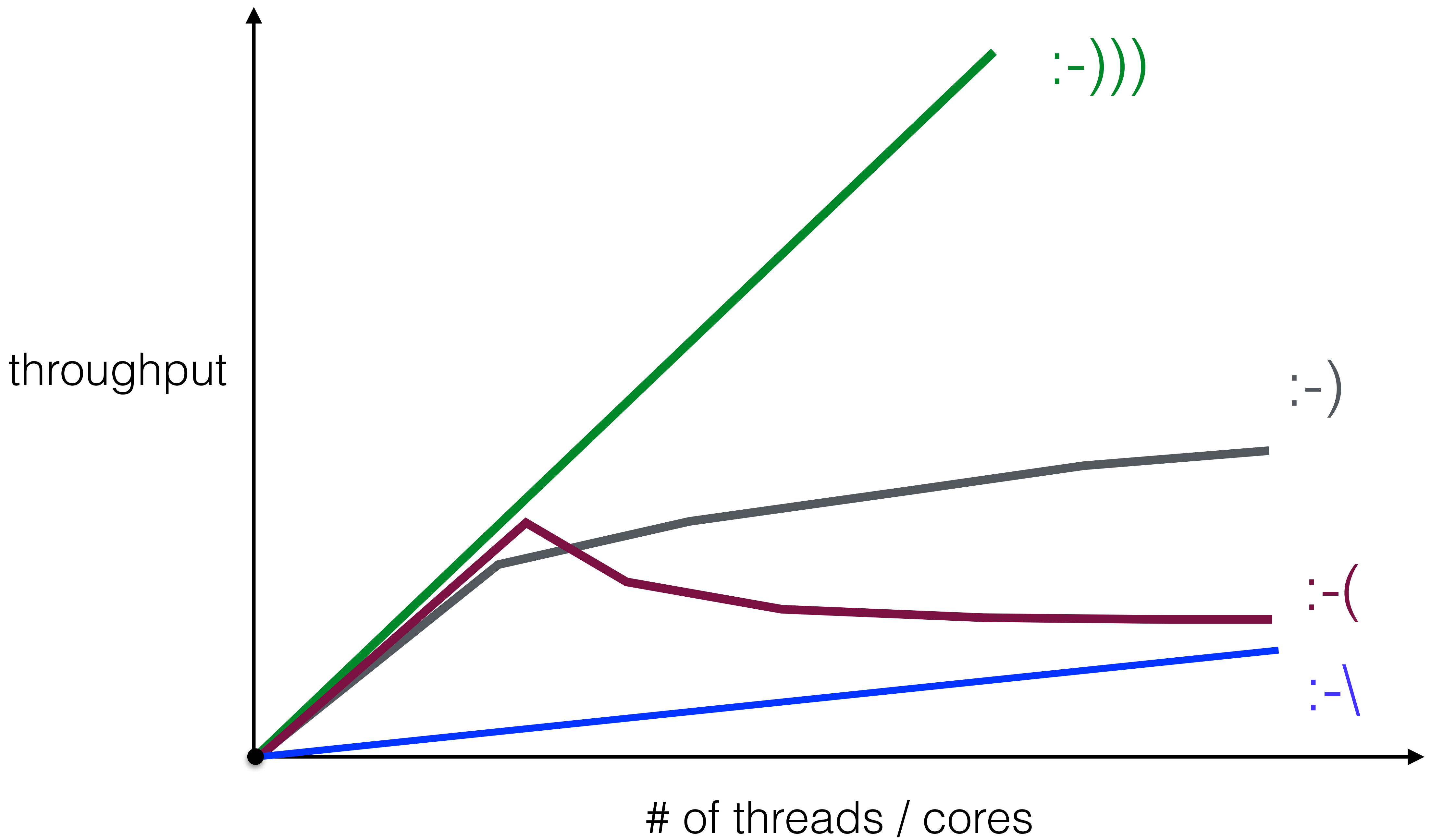


Sequential Consistency [Lamport’79]

there exists a legal sequence that preserves per-thread precedence (program order)



Performance and scalability



Relaxations allow trading

correctness
for
performance

Relaxations allow trading

correctness
for
performance

provide the potential
for better-performing
implementations

Relaxing the Semantics

Relaxing the Semantics

- Sequential specification = set of legal sequences
- Consistency condition = e.g. linearizability / sequential consistency

Relaxing the Semantics

Quantitative relaxations
Henzinger, Kirsch, Payer, Sezgin,S. POPL13

- Sequential specification = set of legal sequences
- Consistency condition = e.g. linearizability / sequential consistency

Relaxing the Semantics

not
“sequentially
correct”

Quantitative relaxations
Henzinger, Kirsch, Payer, Sezgin,S. POPL13

- Sequential specification = set of legal sequences
- Consistency condition = e.g. linearizability / sequential consistency

Relaxing the Semantics

not
“sequentially
correct”

Quantitative relaxations
Henzinger, Kirsch, Payer, Sezgin,S. POPL13

- Sequential specification = set of legal sequences
- Consistency condition = e.g. linearizability / sequential consistency

Local linearizability
in this talk

Relaxing the Semantics

not
"sequentially
correct"

Quantitative relaxations
Henzinger, Kirsch, Payer, Sezgin,S. POPL13

- Sequential specification = set of legal sequences
- Consistency condition = e.g. linearizability / sequential consistency

Local linearizability
in this talk

for queues only
(feel free to ask for more)

Relaxing the Semantics

not
"sequentially
correct"

Quantitative relaxations
Henzinger, Kirsch, Payer, Sezgin,S. POPL13

- Sequential specification = set of legal sequences
- Consistency condition = e.g. linearizability / sequential consistency

Local linearizability
in this talk

for queues only
(feel free to ask for more)

too weak

Relaxing the Consistency Condition

Relaxing the Consistency Condition

Local Linearizability
(CONCUR16)

Local Linearizability
main idea

Local Linearizability main idea

- Partition a history into a set of local histories
- Require linearizability per local history

Local Linearizability main idea

Already present in some shared-memory consistency conditions
(not in our form of choice)

- Partition a history into a set of local histories
- Require linearizability per local history

Local Linearizability main idea

Already present in some shared-memory consistency conditions
(not in our form of choice)

- Partition a history into a set of local histories
- Require linearizability per local history

Local sequential consistency… is also possible

Local Linearizability main idea

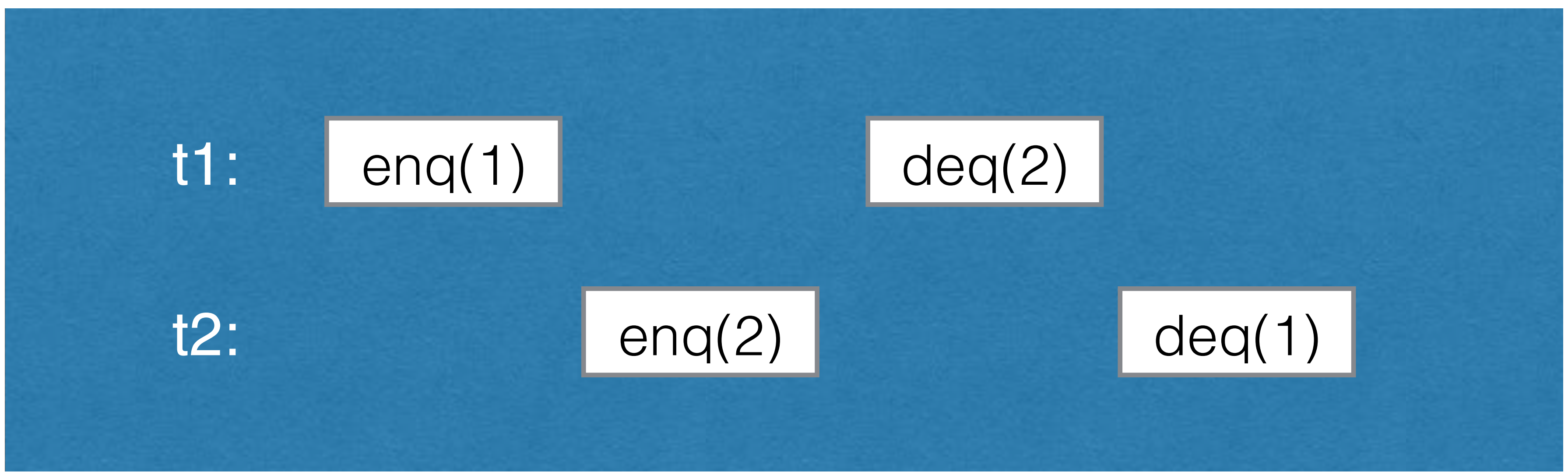
Already present in some shared-memory consistency conditions
(not in our form of choice)

- Partition a history into a set of local histories
- Require linearizability per local history

no global witness

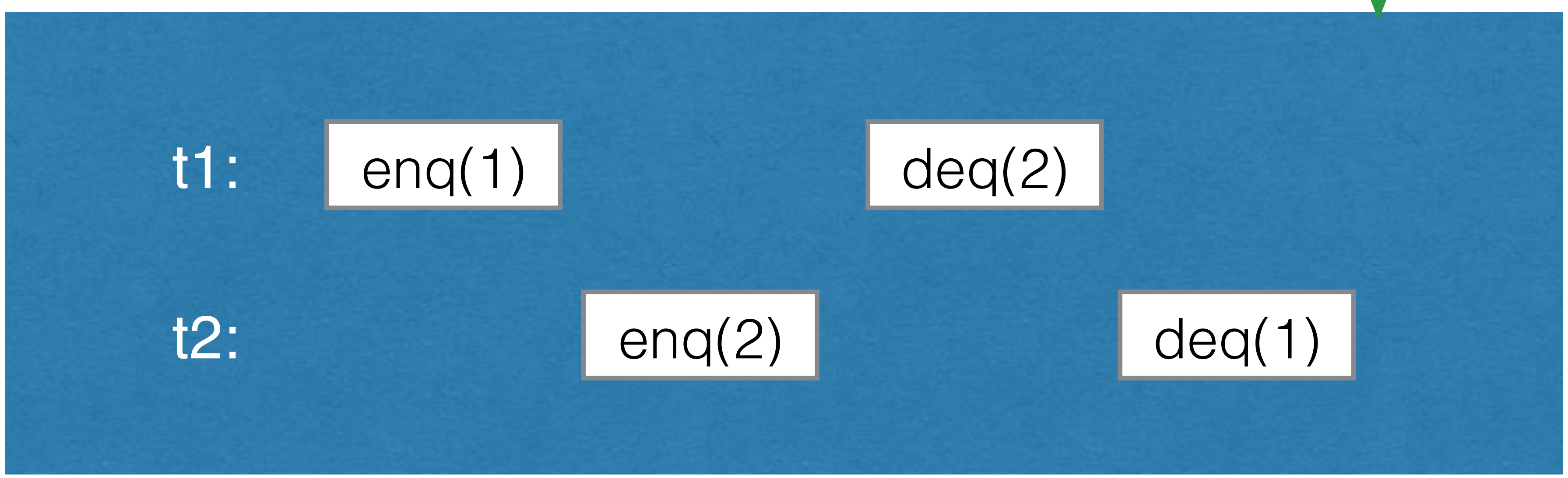
Local sequential consistency… is also possible

Local Linearizability (queue) example

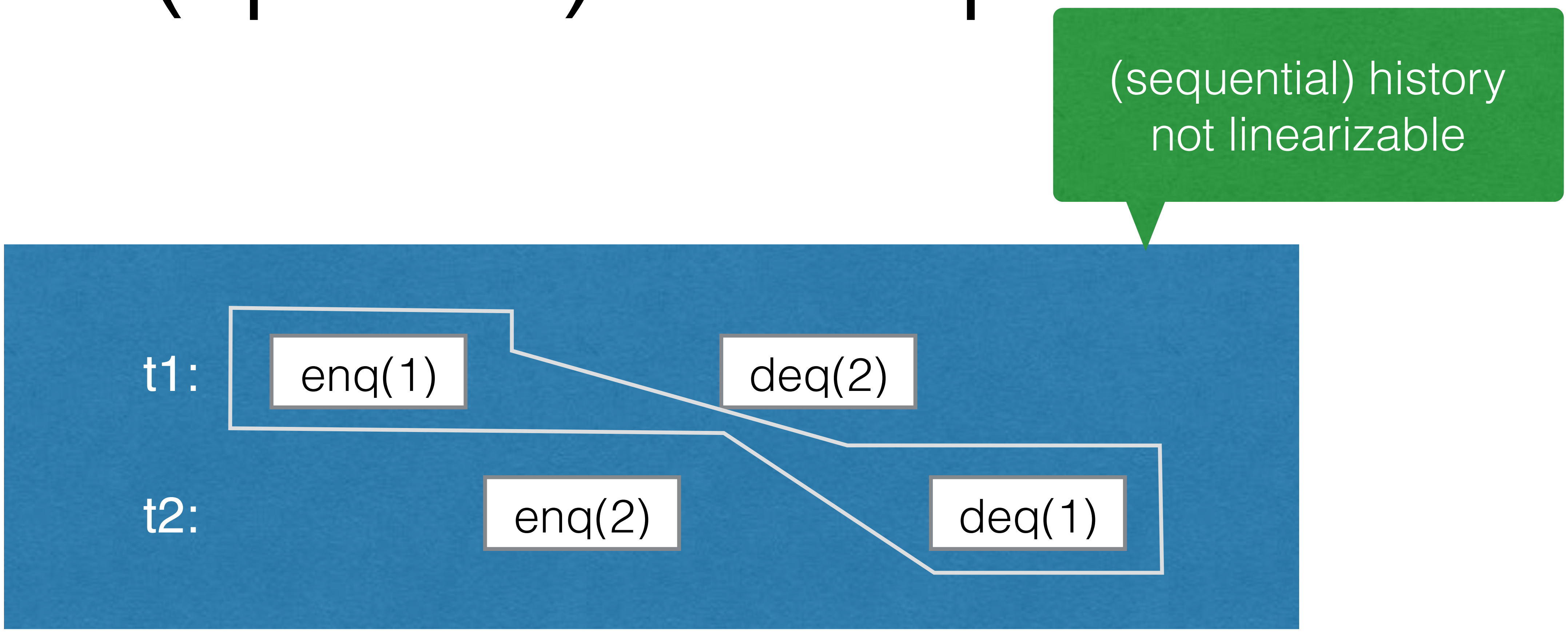


Local Linearizability (queue) example

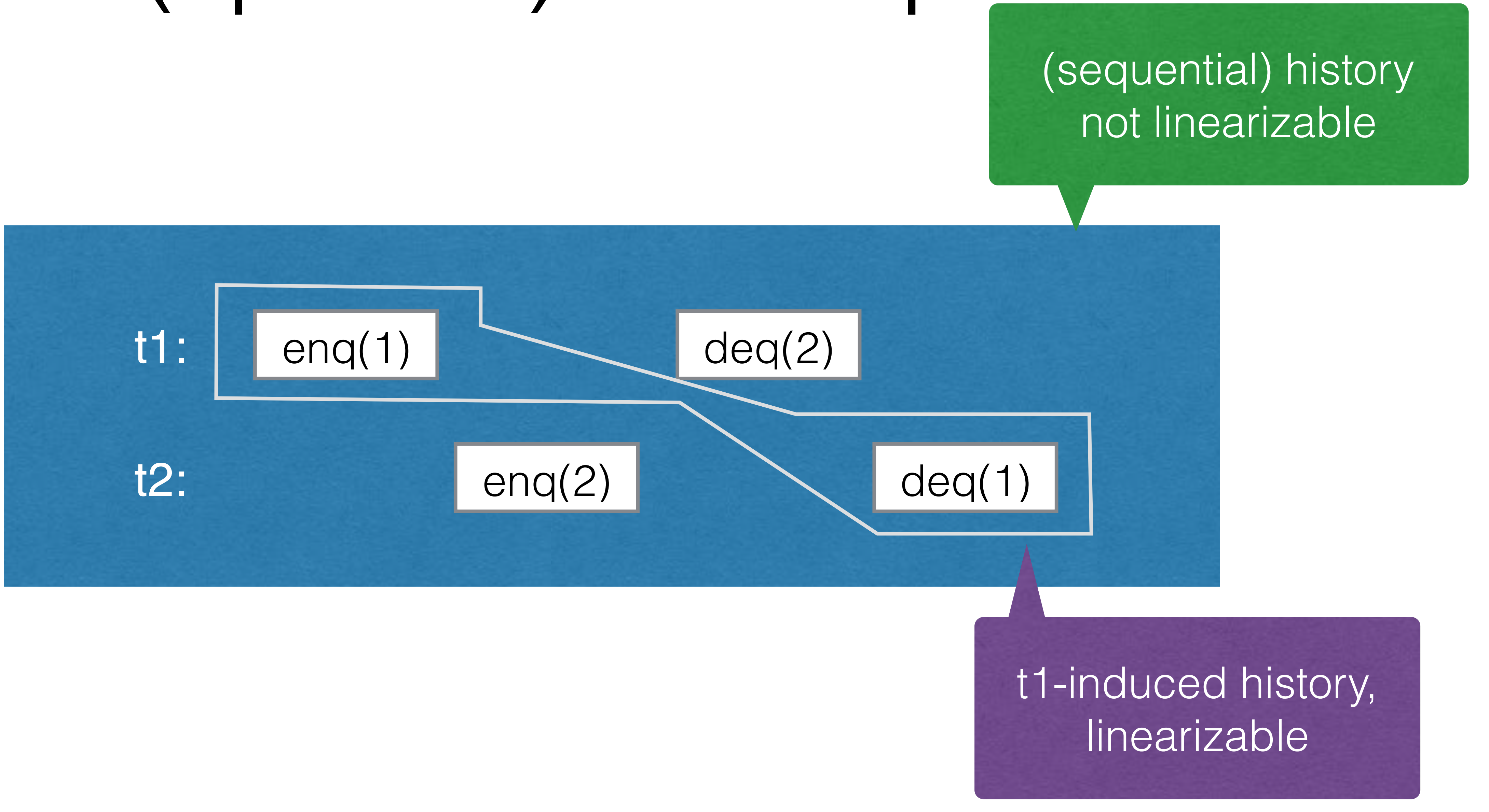
(sequential) history not linearizable



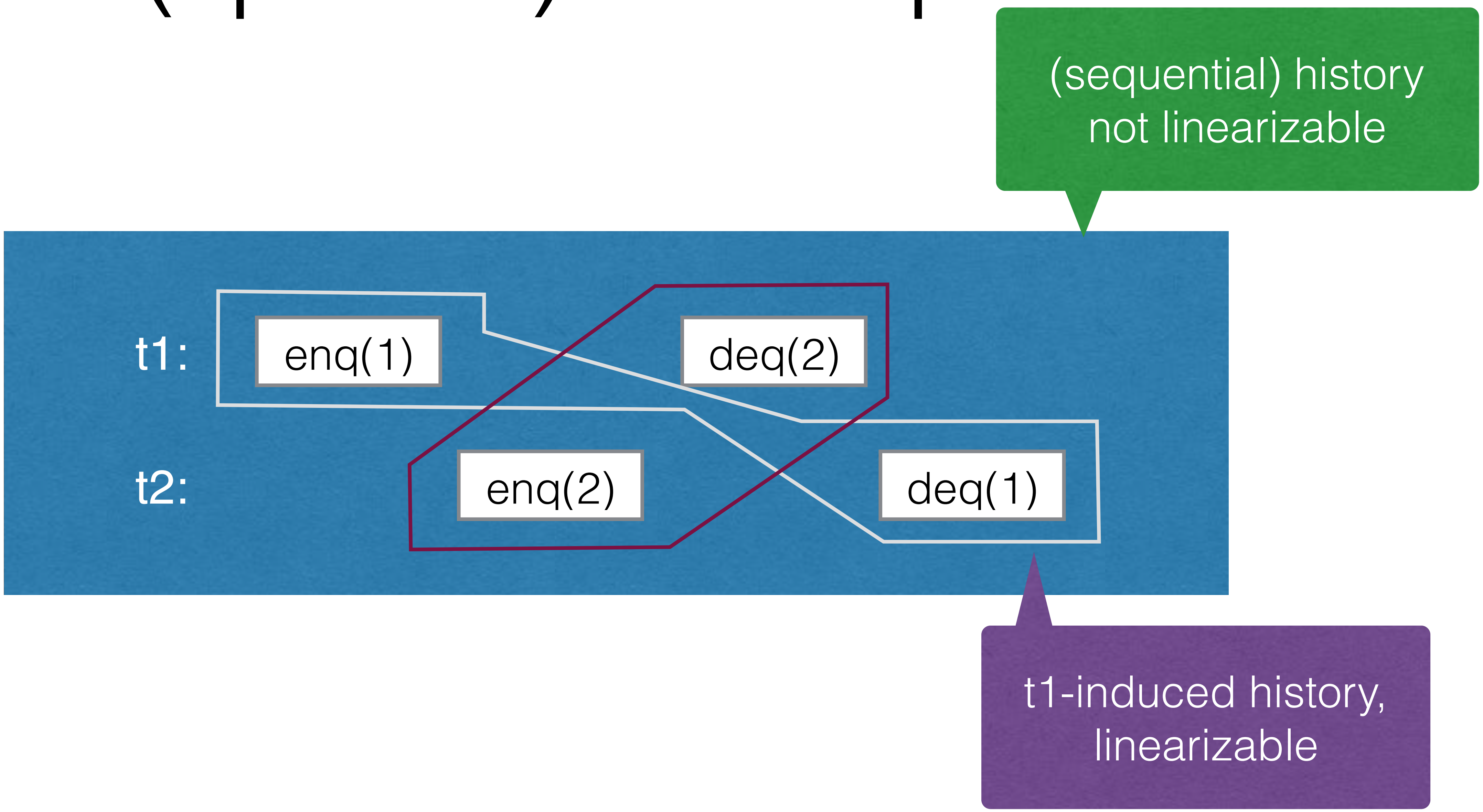
Local Linearizability (queue) example



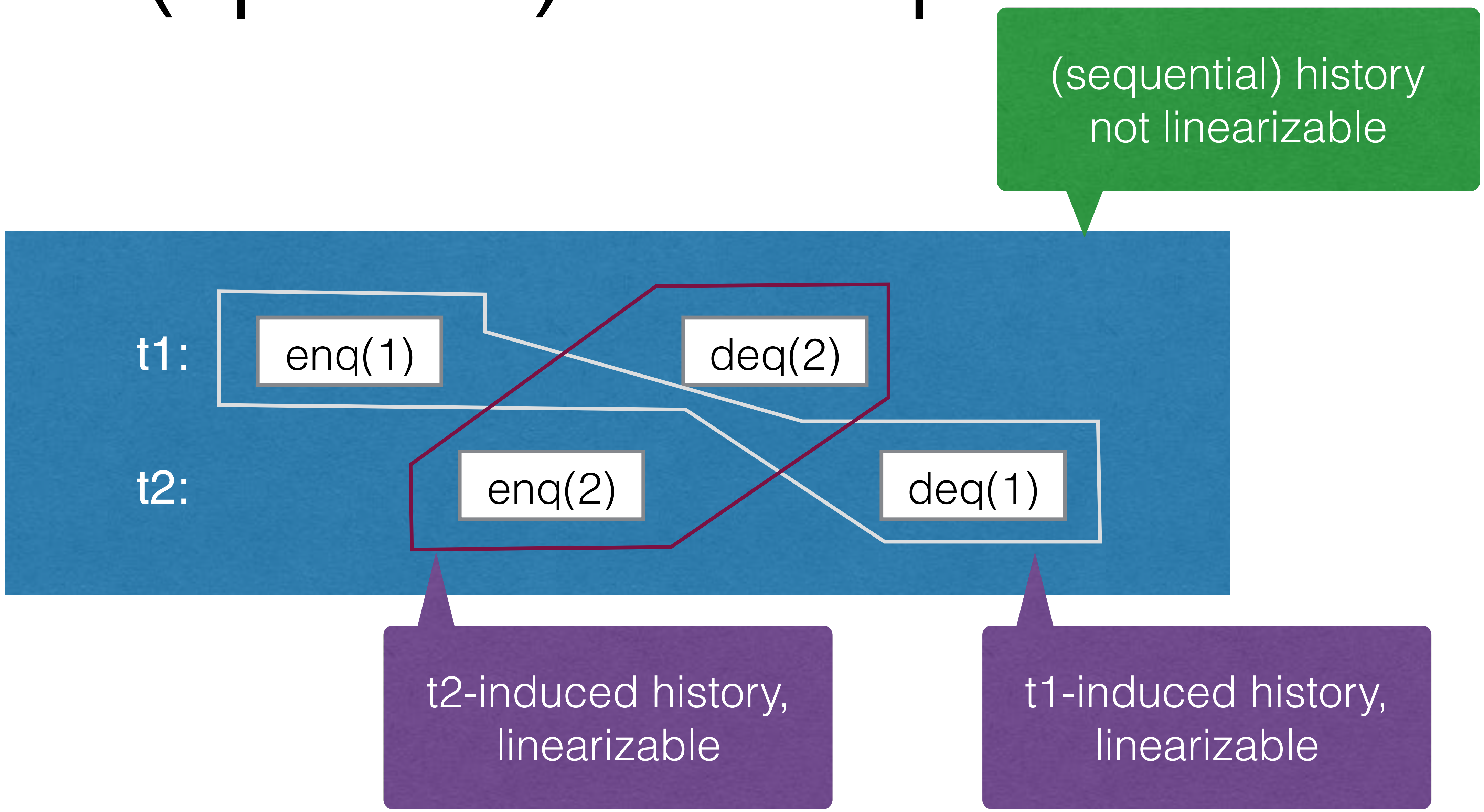
Local Linearizability (queue) example



Local Linearizability (queue) example

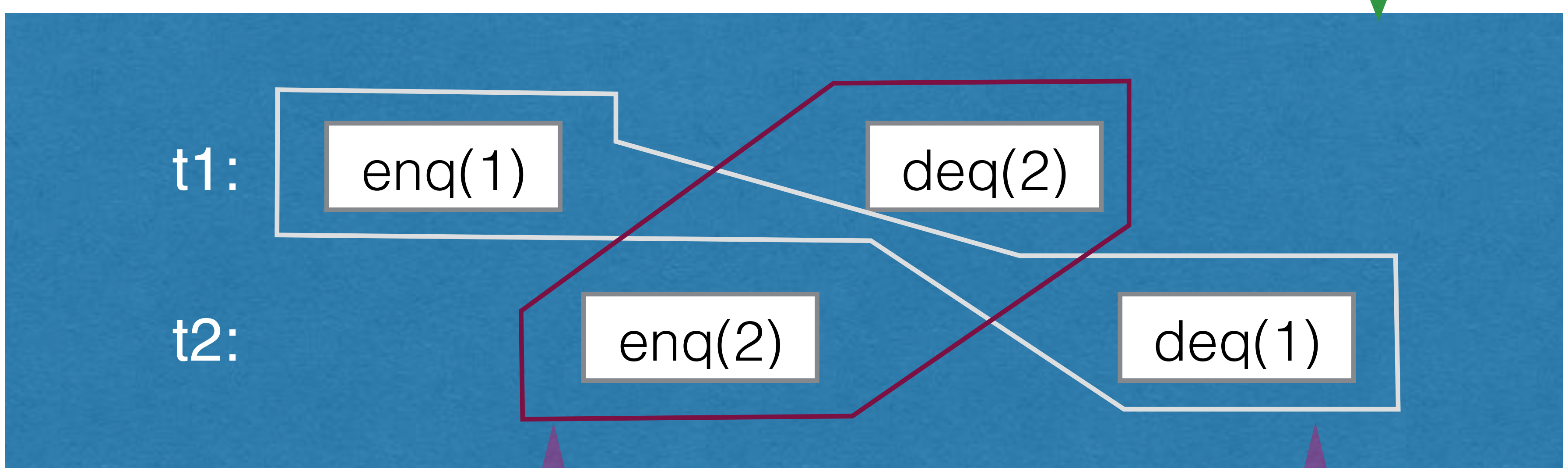


Local Linearizability (queue) example



Local Linearizability (queue) example

(sequential) history not linearizable



t2-induced history, linearizable

t1-induced history, linearizable

locally linearizable

Local Linearizability (queue) definition

Local Linearizability (queue) definition

Queue signature $\Sigma = \{enq(x) \mid x \in V\} \cup \{deq(x) \mid x \in V\} \cup \{deq(empty)\}$

Local Linearizability (queue) definition

Queue signature $\Sigma = \{enq(x) \mid x \in V\} \cup \{deq(x) \mid x \in V\} \cup \{deq(empty)\}$

For a history **h** with a thread T, we put

$$I_T = \{enq(x)^T \in \mathbf{h} \mid x \in V\}$$

$$O_T = \{deq(x)^{T'} \in \mathbf{h} \mid enq(x)^T \in I_T\} \cup \{deq(empty)\}$$

Local Linearizability (queue) definition

Queue signature $\Sigma = \{enq(x) \mid x \in V\} \cup \{deq(x) \mid x \in V\} \cup \{deq(empty)\}$

For a history **h** with a thread T, we put

$I_T = \{enq(x)^T \in \mathbf{h} \mid x \in V\}$

in-methods of thread T are enqueues performed by thread T

$O_T = \{deq(x)^{T'} \in \mathbf{h} \mid enq(x)^T \in I_T\} \cup \{deq(empty)\}$

Local Linearizability (queue) definition

Queue signature $\Sigma = \{enq(x) \mid x \in V\} \cup \{deq(x) \mid x \in V\} \cup \{deq(empty)\}$

For a history **h** with a thread T, we put

$$I_T = \{enq(x)^T \in \mathbf{h} \mid x \in V\}$$

$$O_T = \{deq(x)^{T'} \in \mathbf{h} \mid enq(x)^T \in I_T\} \cup \{deq(empty)\}$$

in-methods of thread T are enqueues performed by thread T

out-methods of thread T are dequeues (performed by any thread) corresponding to enqueues that are in-methods

Local Linearizability (queue) definition

Queue signature $\Sigma = \{enq(x) \mid x \in V\} \cup \{deq(x) \mid x \in V\} \cup \{deq(empty)\}$

For a history **h** with a thread T, we put

$$I_T = \{enq(x)^T \in \mathbf{h} \mid x \in V\}$$

$$O_T = \{deq(x)^{T'} \in \mathbf{h} \mid enq(x)^T \in I_T\} \cup \{deq(empty)\}$$

in-methods of thread T are enqueues performed by thread T

out-methods of thread T are dequeues (performed by any thread) corresponding to enqueues that are in-methods

h is locally linearizable iff every thread-induced history

$$\mathbf{h}_T = \mathbf{h} \mid (I_T \cup O_T)$$

is linearizable.

Generalizations of Local Linearizability

Generalizations of Local Linearizability

Signature Σ

Generalizations of Local Linearizability

Signature Σ

For a history **h** with n threads, choose

$In_{\mathbf{h}}(i)$

$Out_{\mathbf{h}}(i)$

Generalizations of Local Linearizability

Signature Σ

For a history **h** with n threads, choose

$In_{\mathbf{h}}(i)$

in-methods of thread i,
methods that go in $\mathbf{h}_i$

$Out_{\mathbf{h}}(i)$

Generalizations of Local Linearizability

Signature Σ

For a history **h** with n threads, choose

$In_{\mathbf{h}}(i)$

$Out_{\mathbf{h}}(i)$

in-methods of thread i, methods that go in $\mathbf{h}_i$

out-methods of thread i, dependent methods on the methods in $In_{\mathbf{h}}(i)$ (performed by any thread)

Generalizations of Local Linearizability

Signature Σ

For a history **h** with n threads, choose

$In_{\mathbf{h}}(i)$

in-methods of thread i, methods that go in $\mathbf{h}_i$

$Out_{\mathbf{h}}(i)$

out-methods of thread i, dependent methods on the methods in $In_{\mathbf{h}}(i)$ (performed by any thread)

h is locally linearizable iff every thread-induced history

$$\mathbf{h}_i = \mathbf{h} \mid (In_{\mathbf{h}}(i) \cup Out_{\mathbf{h}}(i))$$

is linearizable.

Generalizations of Local Linearizability

Signature Σ

For a history **h** with n threads, choose

$In_{\mathbf{h}}(i)$

in-methods of thread i, methods that go in $\mathbf{h}_i$

$Out_{\mathbf{h}}(i)$

by increasing the in-methods, LL gradually moves to linearizability

out-methods of thread i, dependent methods on the methods in $In_{\mathbf{h}}(i)$ (performed by any thread)

h is locally linearizable iff every thread-induced history

$$\mathbf{h}_i = \mathbf{h} \mid (In_{\mathbf{h}}(i) \cup Out_{\mathbf{h}}(i))$$

is linearizable.

Where do we stand?

In general

Linearizability

Local Linearizability

Sequential Consistency

Where do we stand?

For queues (and most container-type data structures)

Linearizability

Local Linearizability

Sequential Consistency

Properties

Properties

Local linearizability is compositional

Properties

Local linearizability is compositional

like linearizability
unlike sequential consistency

Properties

Local linearizability is compositional

like linearizability
unlike sequential consistency

h (over multiple objects) is locally linearizable
iff
each per-object subhistory of **h** is locally linearizable

Properties

Local linearizability is compositional

like linearizability
unlike sequential consistency

h (over multiple objects) is locally linearizable
iff
each per-object subhistory of **h** is locally linearizable

Local linearizability is modular / "decompositional"

Properties

Local linearizability is compositional

like linearizability
unlike sequential consistency

h (over multiple objects) is locally linearizable
iff
each per-object subhistory of **h** is locally linearizable

Local linearizability is modular / "decompositional"

uses decomposition into smaller histories, by definition

Properties

Local linearizability is compositional

like linearizability
unlike sequential consistency

h (over multiple objects) is locally linearizable
iff
each per-object subhistory of **h** is locally linearizable

Local linearizability is modular / "decompositional"

uses decomposition into smaller histories, by definition

↓

may allow for modular verification

Verification (queue)

Queue sequential specification (axiomatic)

$\mathbf{s}$ is a legal queue sequence

iff

1. $\mathbf{s}$ is a legal pool sequence, and
2. $enq(x) <_{\mathbf{s}} enq(y) \;\wedge\; deq(y) \in \mathbf{s} \quad \Rightarrow \quad deq(x) \in \mathbf{s} \;\wedge\; deq(x) <_{\mathbf{s}} deq(y)$

Verification (queue)

Queue sequential specification (axiomatic)

s is a legal queue sequence

iff

1. **s** is a legal pool sequence, and
2. $enq(x) <_s enq(y) \;\wedge\; deq(y) \in s \;\Rightarrow\; deq(x) \in s \;\wedge\; deq(x) <_s deq(y)$

Queue linearizability (axiomatic)

Henzinger, Sezgin, Vafeiadis CONCUR13

h is queue linearizable

iff

1. **h** is pool linearizable, and
2. $enq(x) <_h enq(y) \;\wedge\; deq(y) \in h \;\Rightarrow\; deq(x) \in h \;\wedge\; deq(y) \not<_h deq(x)$

Verification (queue)

Queue sequential specification (axiomatic)

s is a legal queue sequence

iff

1. **s** is a legal pool sequence, and
2. $enq(x) <_{\mathbf{s}} enq(y) \ \wedge \ deq(y) \in \mathbf{s} \quad \Rightarrow \quad deq(x) \in \mathbf{s} \ \wedge \ deq(x) <_{\mathbf{s}} deq(y)$

Queue linearizability (axiomatic)

Henzinger, Sezgin, Vafeiadis CONCUR13

h is queue linearizable

iff

1. **h** is pool linearizable, and
2. $enq(x) <_{\mathbf{h}} enq(y) \ \wedge \ deq(y) \in \mathbf{h} \quad \Rightarrow \quad deq(x) \in \mathbf{h} \ \wedge \ deq(y) \not<_{\mathbf{h}} deq(x)$

precedence order

Verification (queue)

Queue sequential specification (axiomatic)

s is a legal queue sequence

iff

1. **s** is a legal pool sequence, and
2. $enq(x) <_{\mathbf{s}} enq(y) \ \wedge \ deq(y) \in \mathbf{s} \quad \Rightarrow \quad deq(x) \in \mathbf{s} \ \wedge \ deq(x) <_{\mathbf{s}} deq(y)$

Queue local linearizability (axiomatic)

h is queue locally linearizable

iff

1. **h** is pool locally linearizable, and
2. $enq(x) <_{\mathbf{h}}^{i} enq(y) \ \wedge \ deq(y) \in \mathbf{h} \quad \Rightarrow \quad deq(x) \in \mathbf{h} \ \wedge \ deq(y) \not<_{\mathbf{h}} deq(x)$

Verification (queue)

Queue sequential specification (axiomatic)

s is a legal queue sequence

iff

1. **s** is a legal pool sequence, and
2. $enq(x) <_{\mathbf{s}} enq(y) \;\wedge\; deq(y) \in \mathbf{s} \quad \Rightarrow \quad deq(x) \in \mathbf{s} \;\wedge\; deq(x) <_{\mathbf{s}} deq(y)$

Queue local linearizability (axiomatic)

h is queue locally linearizable

iff

1. **h** is pool locally linearizable, and
2. $enq(x) <^{i}_{\mathbf{h}} enq(y) \;\wedge\; deq(y) \in \mathbf{h} \quad \Rightarrow \quad deq(x) \in \mathbf{h} \;\wedge\; deq(y) \not<_{\mathbf{h}} deq(x)$

thread-local precedence order

Generic Implementations

Generic Implementations

Your favorite linearizable data structure implementation

Generic Implementations

Your favorite linearizable data structure implementation

Φ

Generic Implementations

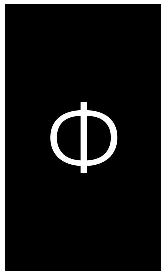
Your favorite linearizable data structure implementation

Φ

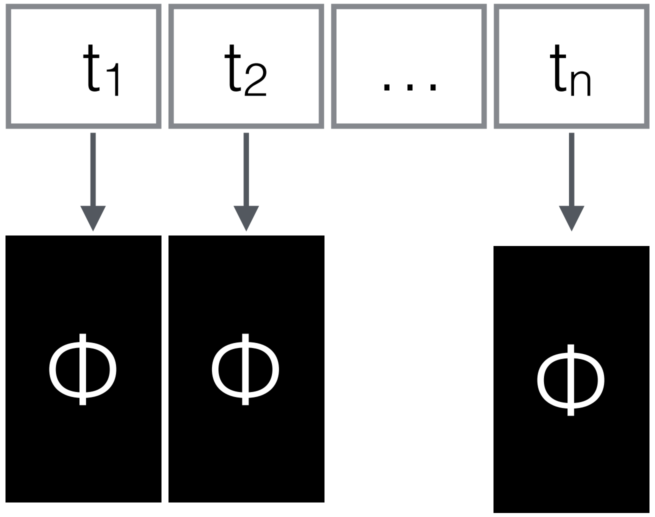
turns into a locally linearizable implementation by:

Generic Implementations

Your favorite linearizable data structure implementation

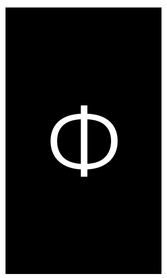


turns into a locally linearizable implementation by:

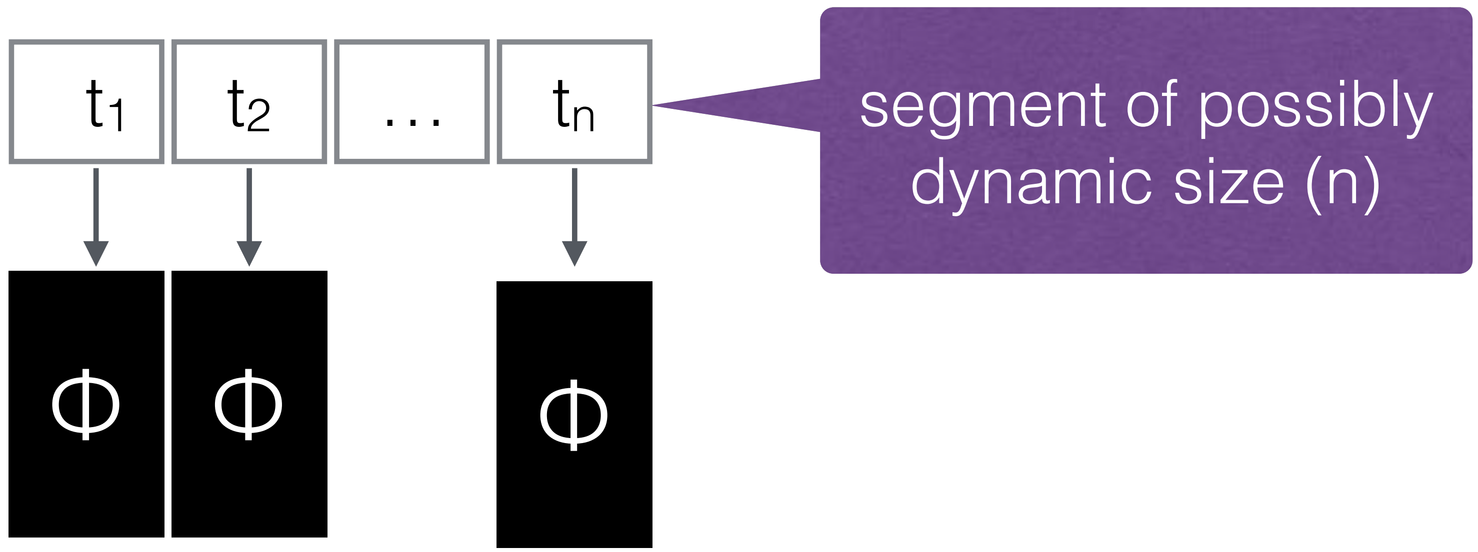


Generic Implementations

Your favorite linearizable data structure implementation



turns into a locally linearizable implementation by:

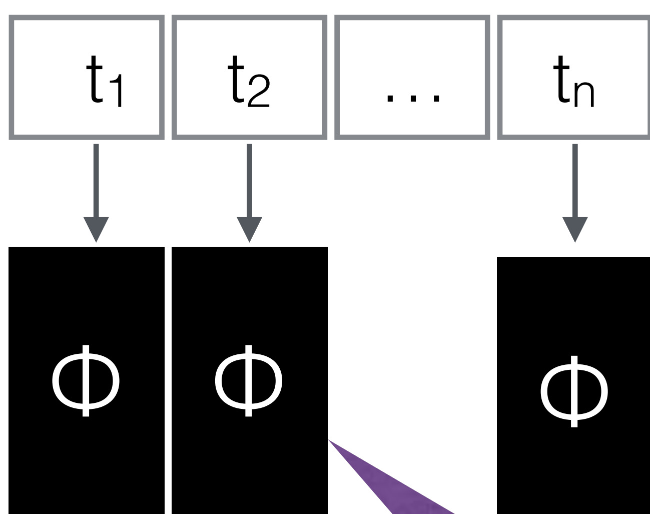


Generic Implementations

Your favorite linearizable data structure implementation

Φ

turns into a locally linearizable implementation by:

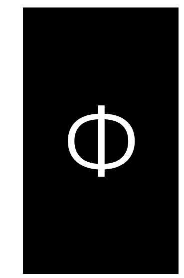


segment of possibly dynamic size (n)

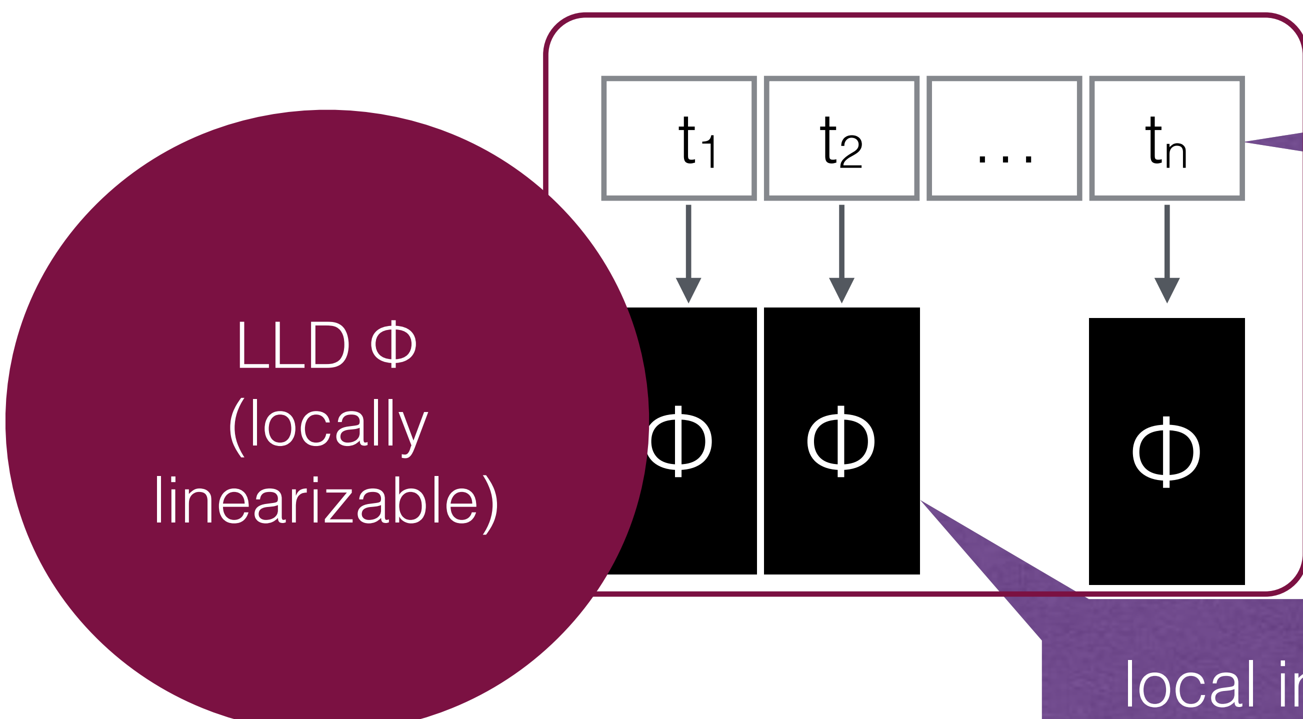
local inserts / global (randomly distributed) removes

Generic Implementations

Your favorite linearizable data structure implementation



turns into a locally linearizable implementation by:



segment of possibly dynamic size (n)

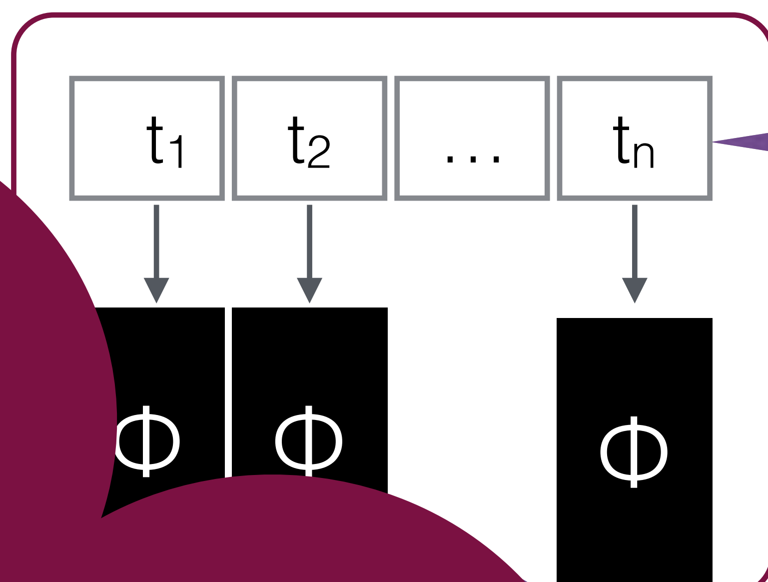
local inserts / global (randomly distributed) removes

Generic Implementations

Your favorite linearizable data structure implementation

Φ

turns into a locally linearizable implementation by:



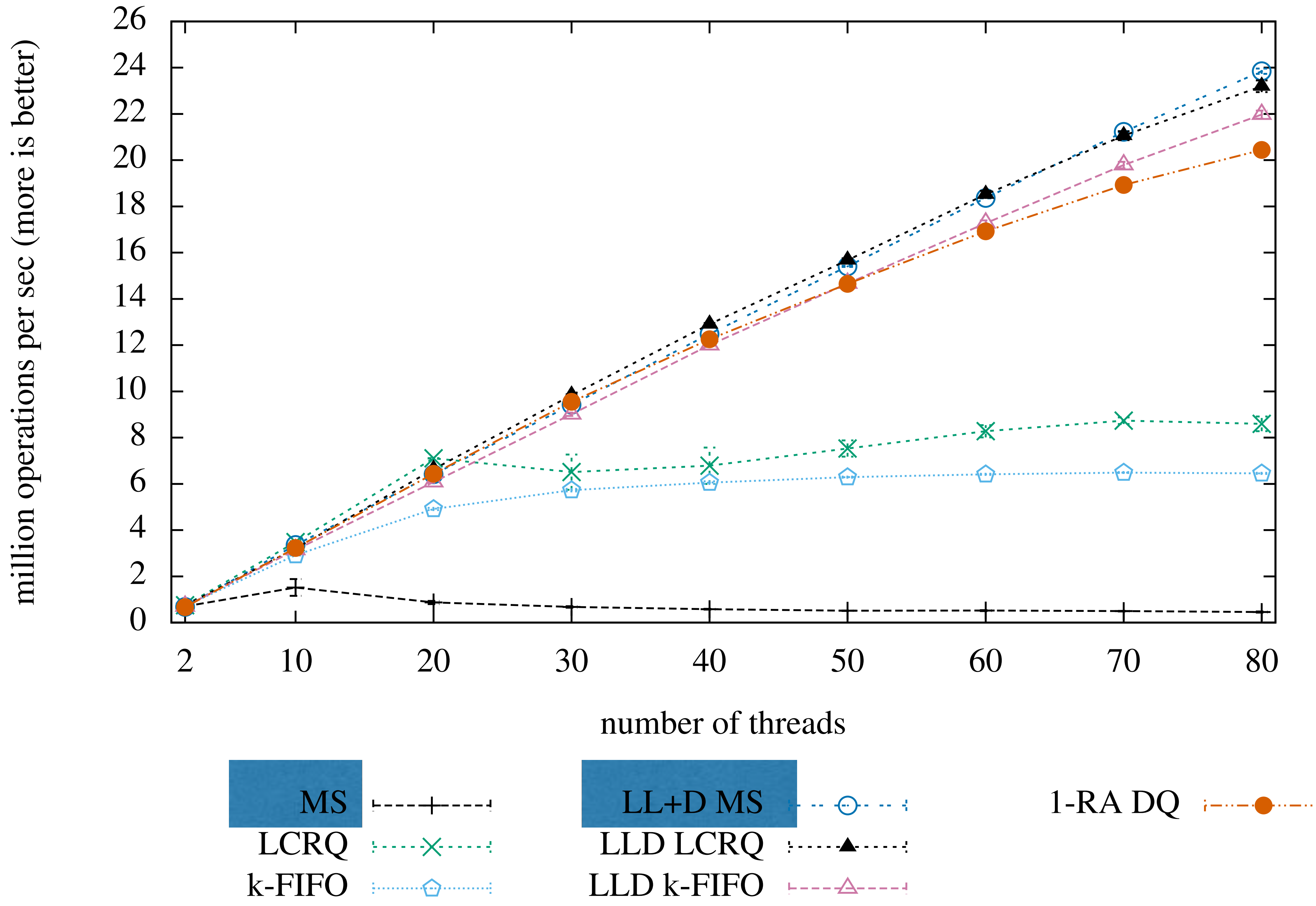
segment of possibly dynamic size (n)

local inserts / global (randomly distributed) removes

LLD Φ (locally linearizable)

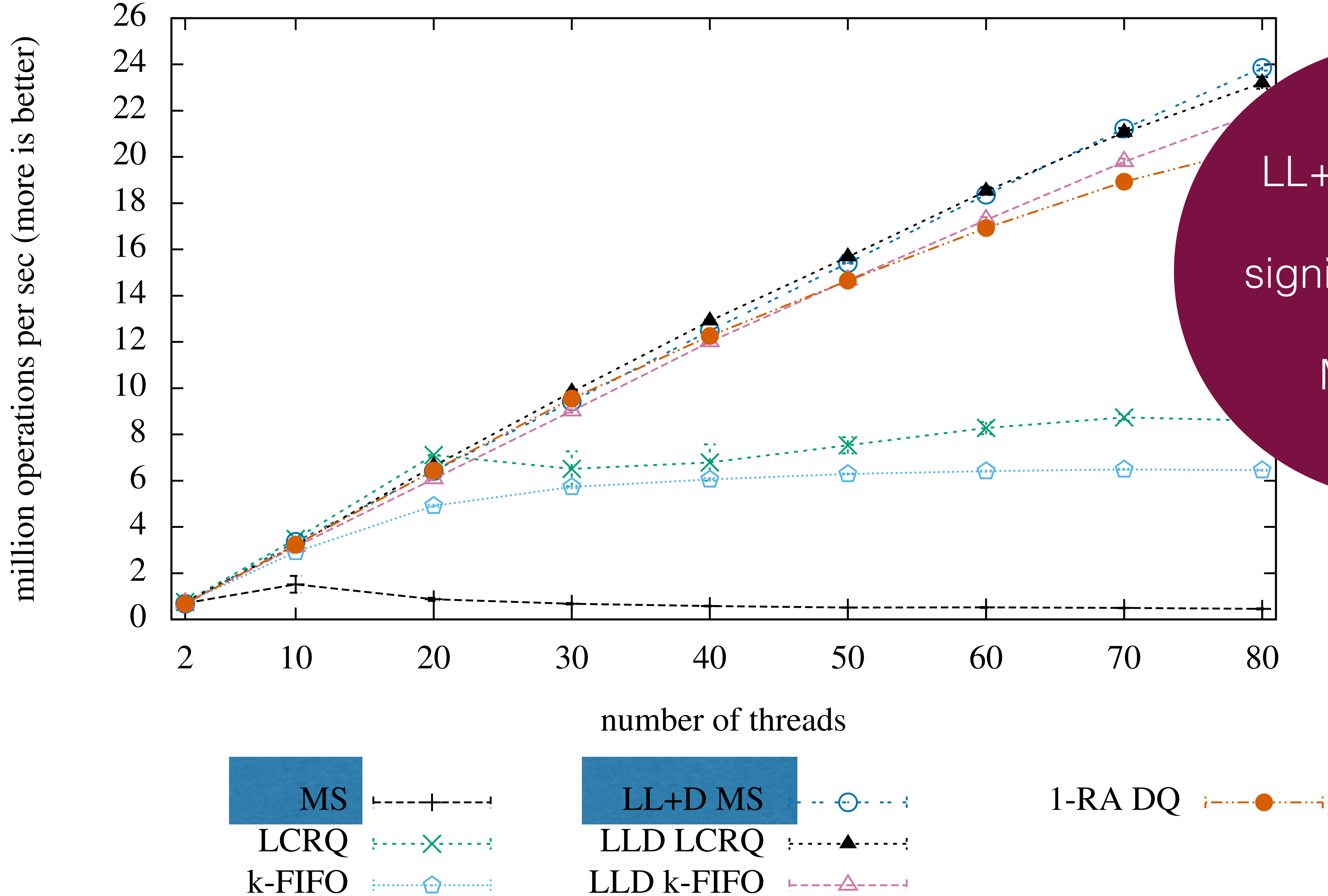
LL+D Φ (also pool linearizable)

Performance



(a) Queues, LL queues, and "queue-like" pools

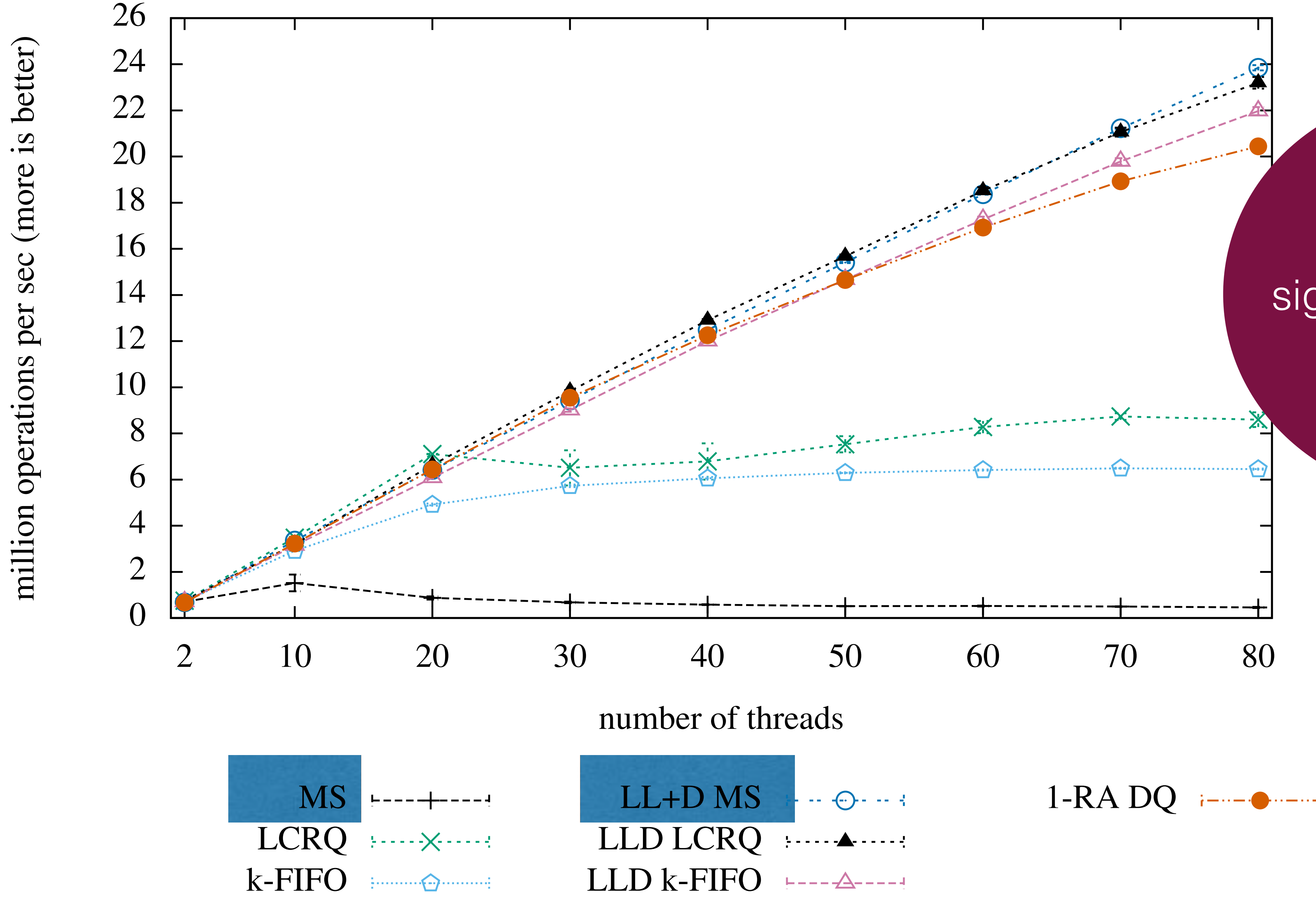
Performance



(a) Queues, LL queues, and "queue-like" pools

LL+D MS queue performs significantly better than MS queue

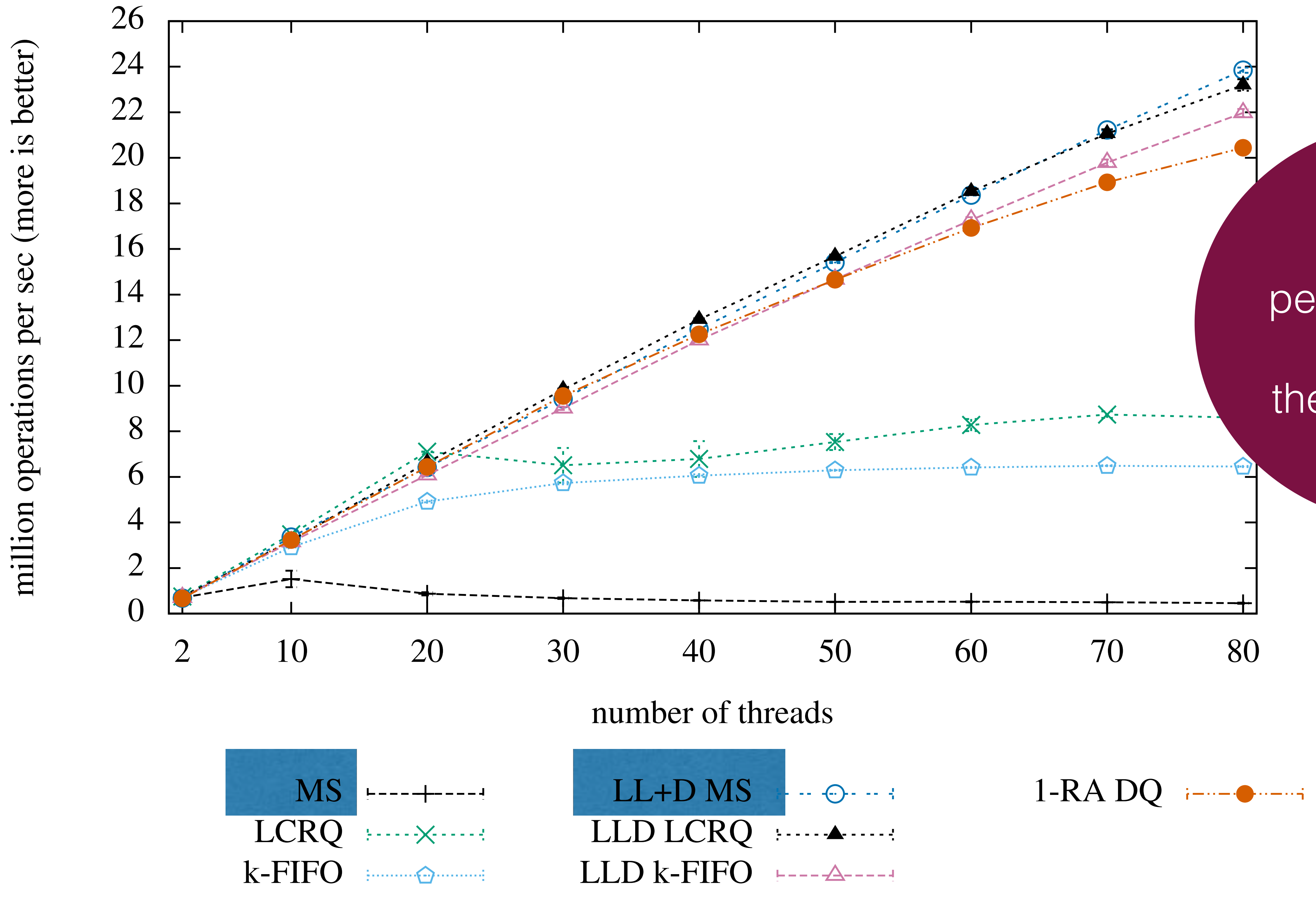
Performance



(a) Queues, LL queues, and “queue-like” pools

LLD Φ
performs
significantly better
than
Φ

Performance



(a) Queues, LL queues, and "queue-like" pools

LL+D MS queue performs better than the best known pools

Thank You!

Local Linearizability

Ana Sokolova UNIVERSITY of SALZBURG

joint work with:

Andreas Haas Google

Andreas Holzer UNIVERSITY OF TORONTO

Michael Lippautz Google

Ali Sezgin UNIVERSITY OF CAMBRIDGE

Tom Henzinger IST AUSTRIA

Christoph Kirsch UNIVERSITY of SALZBURG

Hannes Payer Google

Helmut Veith TU WIEN

Thank You!

Local Linearizability

Ana Sokolova UNIVERSITY of SALZBURG

joint work with:

Andreas Haas Google

Andreas Holzer UNIVERSITY OF TORONTO

Michael Lippautz Google

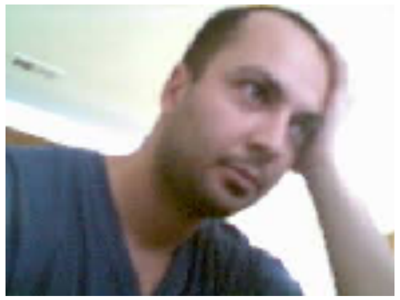
Ali Sezgin UNIVERSITY OF CAMBRIDGE

Tom Henzinger IST AUSTRIA

Christoph Kirsch UNIVERSITY of SALZBURG

Hannes Payer Google

Helmut Veith TU WIEN

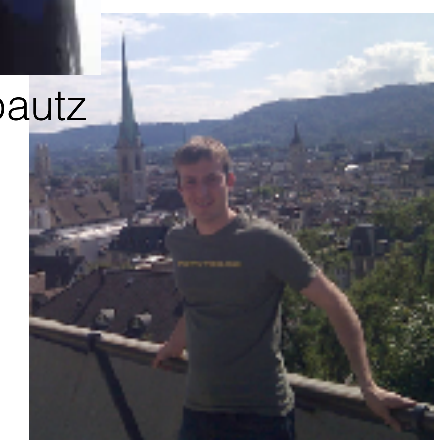


Ali Sezgin UNIVERSITY OF CAMBRIDGE

Hannes Payer Google

Andreas Holzer Google UNIVERSITY OF TORONTO

Michael Lippautz



Andreas Haas Google

Tom Henzinger IST AUSTRIA

Helmut Veith TU WIEN

Christoph Kirsch UNIVERSITY of SALZBURG