Synthesizing Concurrent Schedulers for Irregular Algorithms

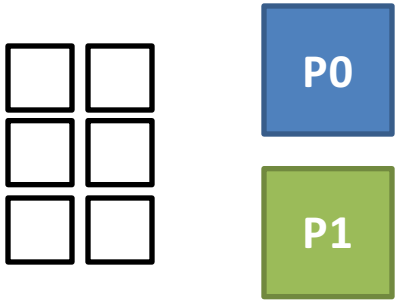
Donald Nguyen and Keshav Pingali

University of Texas at Austin

The Scheduling Problem

- Given a decomposition of program into tasks or *activities*, determine
  a) the assignment of activities to processors **and**
  b) the order in which each processor executes its activities
- In this work, focus on dynamic scheduling

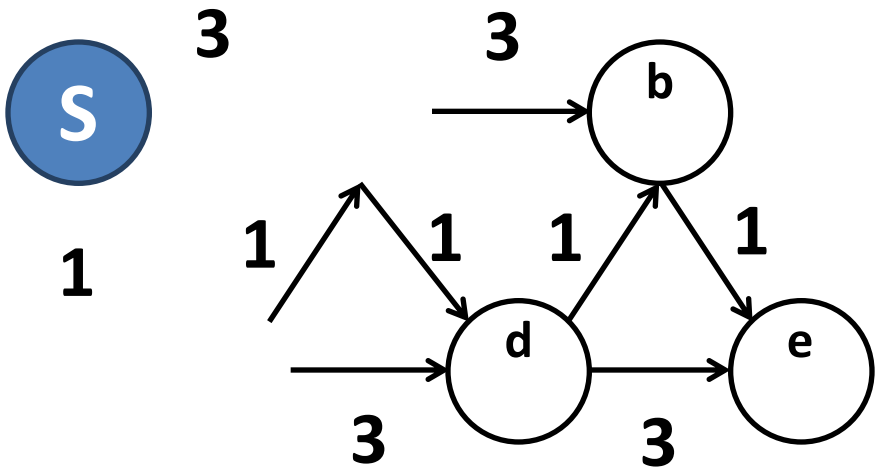
The Scheduling Problem



- Given a decomposition of program into tasks or *activities*, determine
  a) the assignment of activities to processors **and**
  b) the order in which each processor executes its activities
- In this work, focus on dynamic scheduling

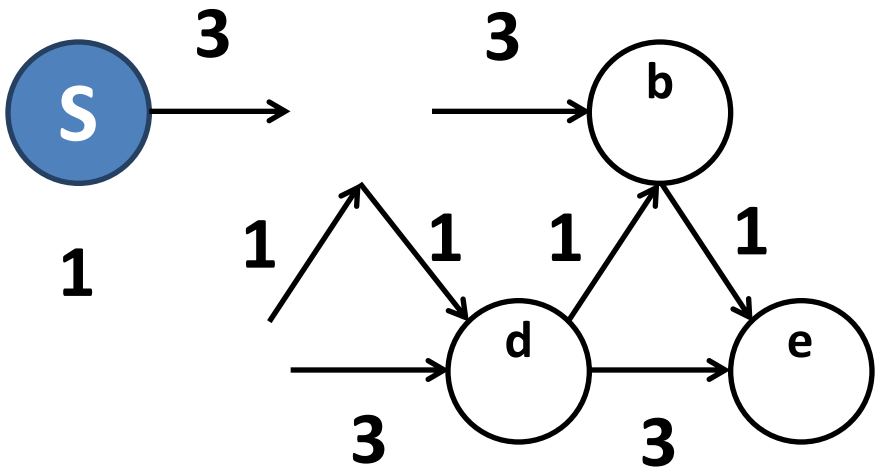
Scheduling Is Important

- Find the shortest path from source to all nodes in graph via iterative relaxation



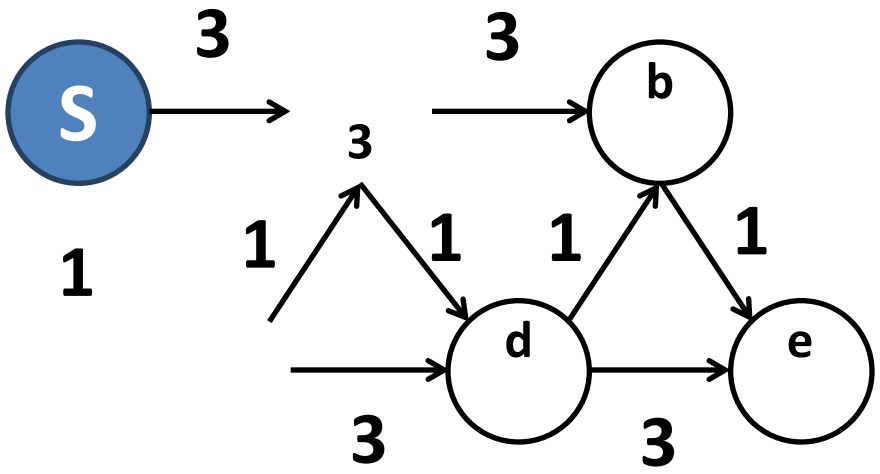
Scheduling Is Important

- Find the shortest path from source to all nodes in graph via iterative relaxation



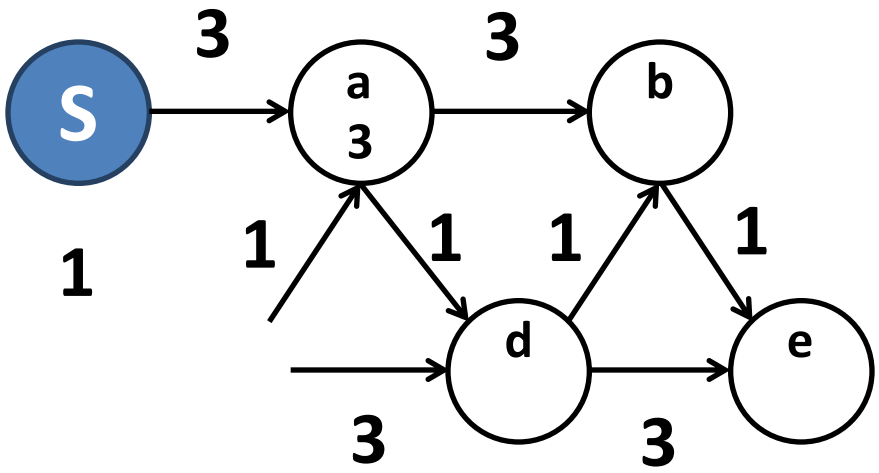
Scheduling Is Important

- Find the shortest path from source to all nodes in graph via iterative relaxation



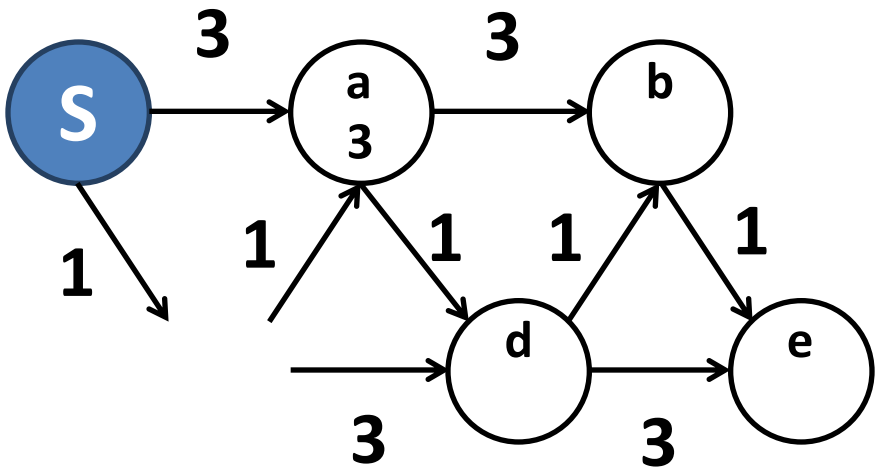
Scheduling Is Important

- Find the shortest path from source to all nodes in graph via iterative relaxation



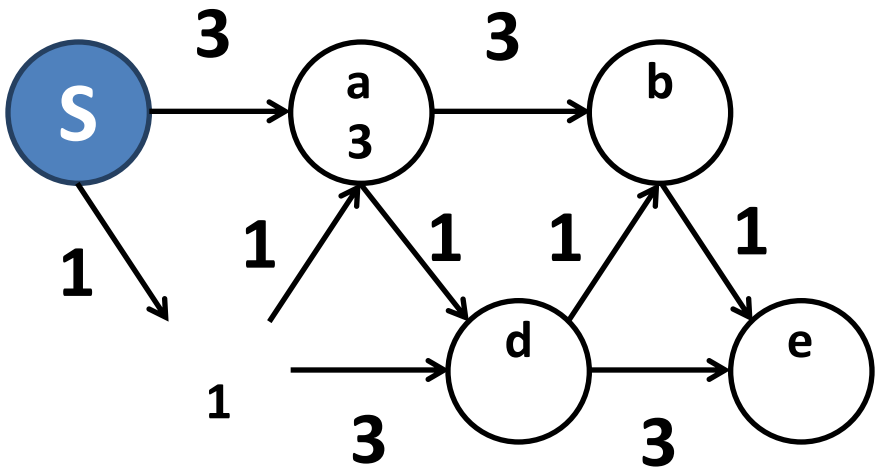
Scheduling Is Important

- Find the shortest path from source to all nodes in graph via iterative relaxation



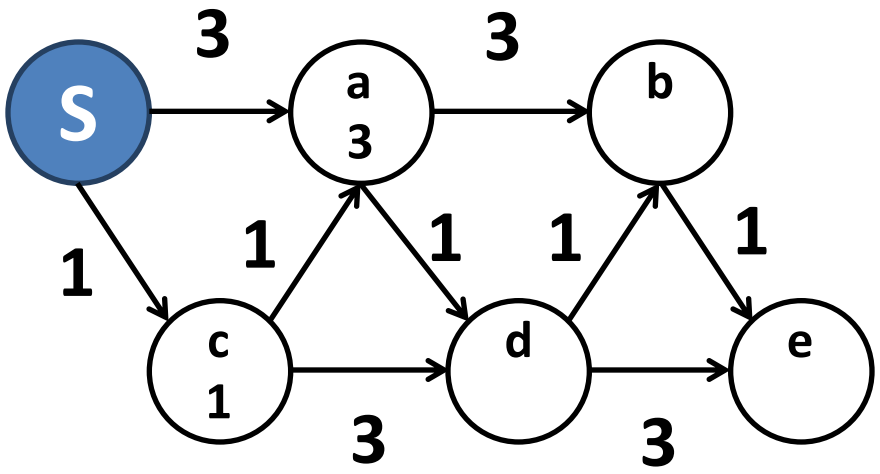
Scheduling Is Important

- Find the shortest path from source to all nodes in graph via iterative relaxation



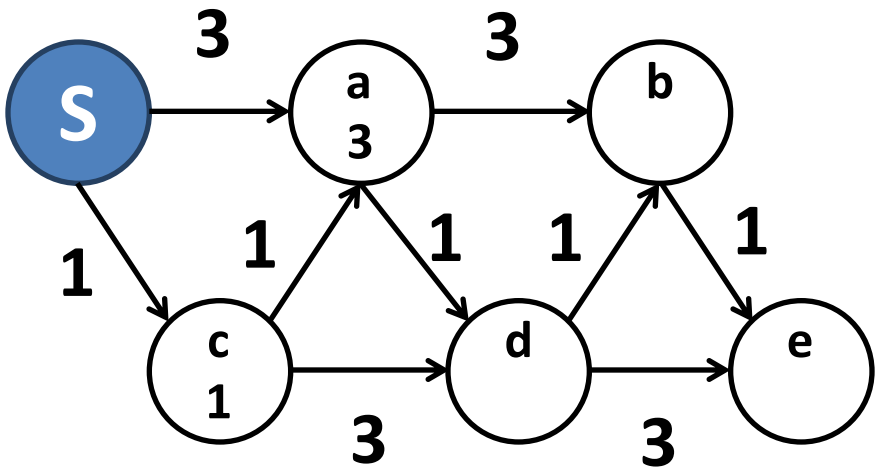
Scheduling Is Important

- Find the shortest path from source to all nodes in graph via iterative relaxation



Scheduling Is Important

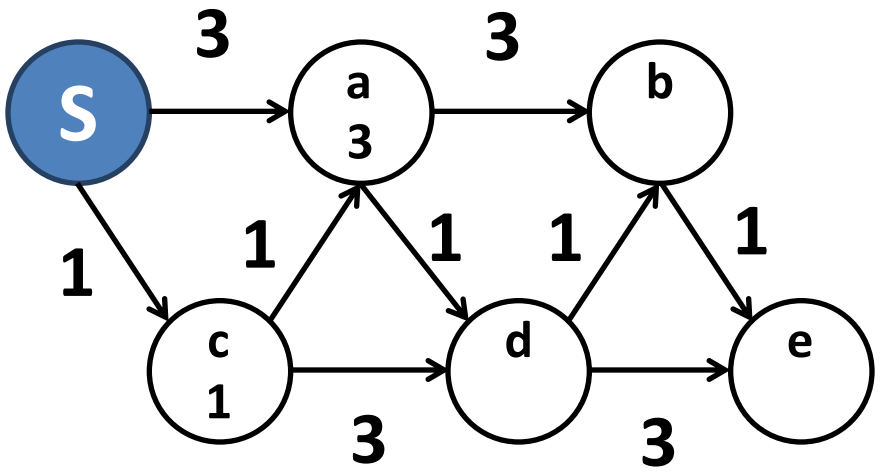
- Find the shortest path from source to all nodes in graph via iterative relaxation



- Policies
  - Distance Order, FIFO, D-stepping

Scheduling Is Important

- Find the shortest path from source to all nodes in graph via iterative relaxation

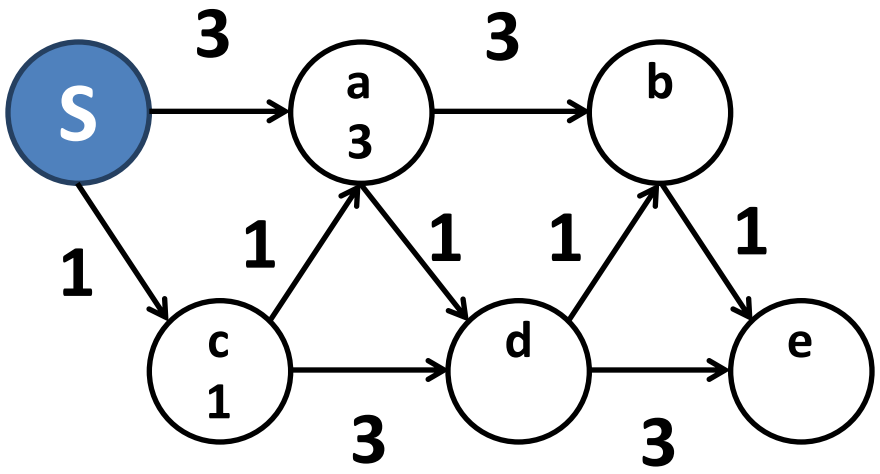


- Policies
  - Distance Order, FIFO, D-stepping

	t=1
Dist. order	0.5 s

Scheduling Is Important

- Find the shortest path from source to all nodes in graph via iterative relaxation

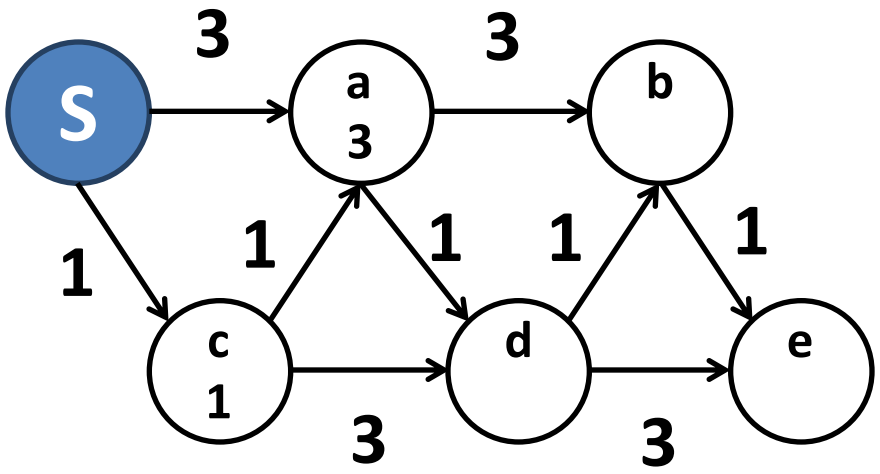


- Policies
  - Distance Order, FIFO, Δ-stepping

	t=1
Dist. order	0.5 s
FIFO	**> 2.5 hours**

Scheduling Is Important

- Find the shortest path from source to all nodes in graph via iterative relaxation

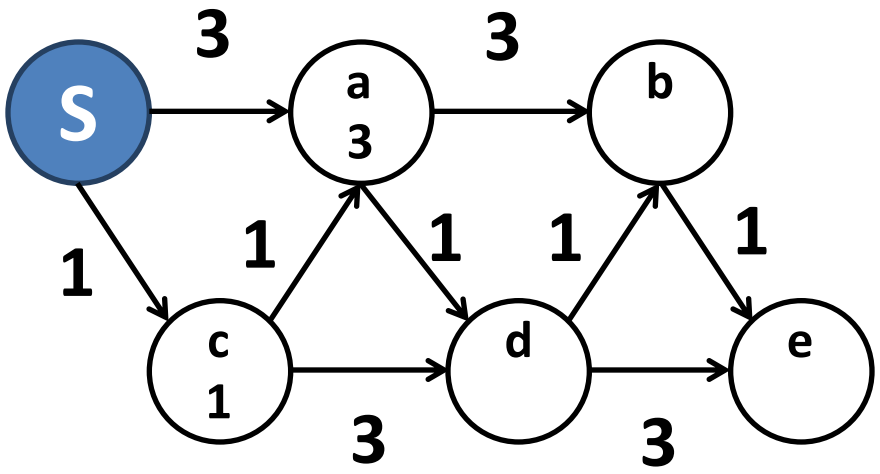


- Policies
  - Distance Order, FIFO, Δ-stepping

	t=1	t=4
Dist. order	0.5 s	20 s
FIFO	**> 2.5 hours**	**~ 1 hour**

Scheduling Is Important

- Find the shortest path from source to all nodes in graph via iterative relaxation

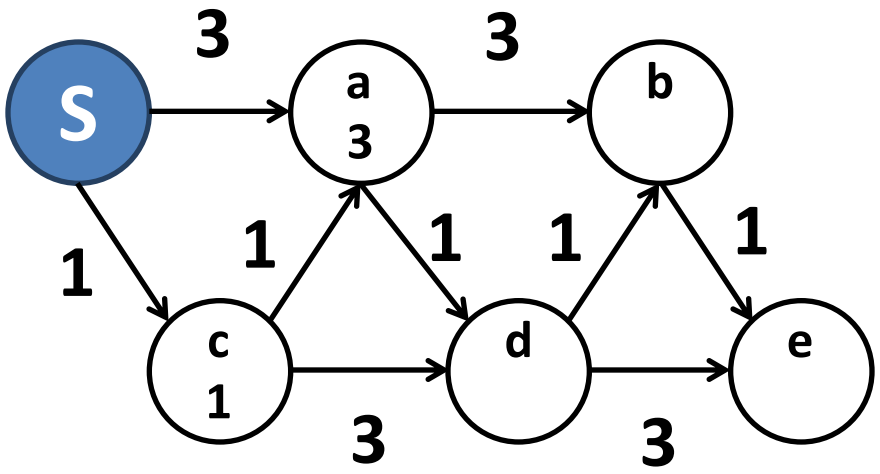


- Policies
  - Distance Order, FIFO, D-stepping

	t=1	t=4
Dist. order	0.5 s	20 s
FIFO	**> 2.5 hours**	**~ 1 hour**
D-stepping	0.5 s	0.4 s

Scheduling Is Important

- Find the shortest path from source to all nodes in graph via iterative relaxation



- Policies
  - Distance Order, FIFO, D-stepping

	t=1	t=4
Dist. order	0.5 s	20 s
FIFO	**> 2.5 hours**	**~ 1 hour**
D-stepping	0.5 s	0.4 s

Order matters even in unordered algorithms

Scheduling Is Hard

Sequential Scheduling

```
while (item = wl.poll())
  fn(item)
  wl.add(item)
```

Scheduling Is Hard

Sequential Scheduling

```
while (item = wl.poll())
  fn(item)
  wl.add(item)
```

Tedious when exploring new policies

Scheduling Is Hard

Sequential Scheduling

```
while (item = wl.poll())
  fn(item)
  wl.add(item)
```

Parallel Scheduling

```
foreach (item : S)
  fn(item)
```

```
spawn A, B, C
sync
```

Tedious when exploring new policies

Scheduling Is Hard

Sequential Scheduling

```
while (item = wl.poll())
  fn(item)
  wl.add(item)
```

Tedious when exploring new policies

Parallel Scheduling

```
foreach (item : S)
  fn(item)
```

```
spawn A, B, C
sync
```

At most, *k* scheduling policies (*k* ≥ 2)

Scheduling Is Hard

Sequential Scheduling

```
while (item = wl.poll())
  fn(item)
  wl.add(item)
```

Tedious when exploring new policies

Parallel Scheduling

```
foreach (item : S)
  fn(item)
```

```
spawn A, B, C
sync
```

OpenMP ($k = 0$)
.Net TPL ($k = 1$)
TBB ($k = 2$)

At most, k scheduling policies ($k \geq 2$)

Scheduling Is Hard

Sequential Scheduling

```
while (item = wl.poll())
  fn(item)
  wl.add(item)
```

Tedious when exploring new policies

Parallel Scheduling

```
foreach (item : S)
  fn(item)
```

```
spawn A, B, C
sync
```

OpenMP ($k = 0$)
.Net TPL ($k = 1$)
TBB ($k = 2$)

At most, k scheduling policies ($k \geq 2$)

New policies *same as* making concurrent, scalable data structures

Scheduling Is Hard

Sequential Scheduling

```
while (item = wl.poll())
  fn(item)
  wl.add(item)
```

Tedious when exploring new policies

Parallel Scheduling

```
foreach (item : S)
  fn(item)
```

```
spawn A, B, C
sync
```

OpenMP ($k = 0$)
.Net TPL ($k = 1$)
TBB ($k = 2$)

At most, k scheduling policies ($k \geq 2$)

New policies *same as* making concurrent, scalable data structures

GaloisRuntime.*foreach*(S, fn, *SchedulingPolicy*);

Contributions

- A language for scheduling policies
  - *Declarative:* sophisticated schedulers w/o writing code
  - *Effective:* performance comparable to hand-written and often better than previous schedulers

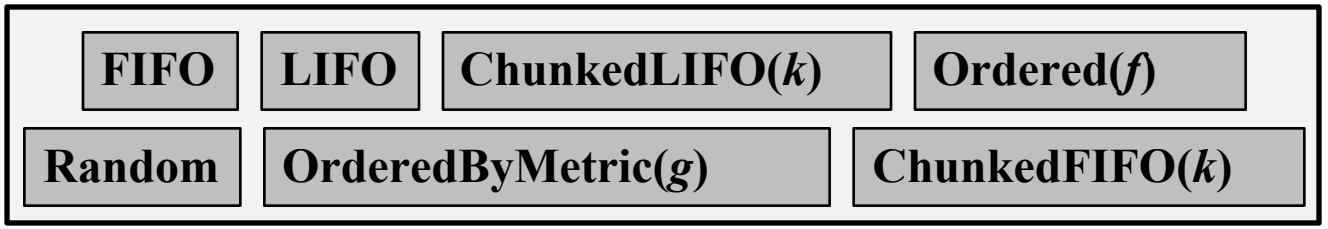
Get good performance without writing (serial or concurrent) scheduling code

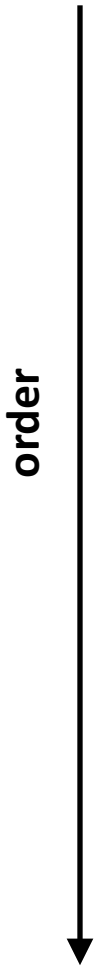
Outline

1. **Language**
2. Synthesis
3. Results

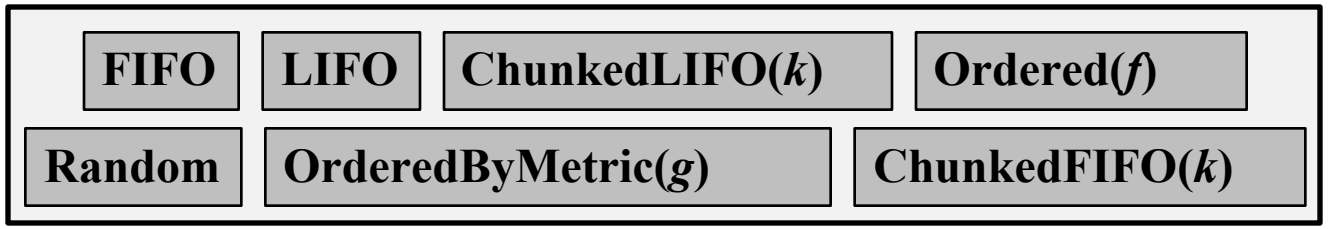
P	::=	Global:D Local:D	*Specification*
D	::=	$R_{\mathrm{NF}}*$ $R_{\mathrm{F}}?$	*Ordering*
R_{NF}	::=	ChunkedFIFO(k)	*Non-final rule*
	|	ChunkedLIFO(k)	
	|	Ordered($f : x, y \to \mathrm{bool}$)	
	|	OrderedByMetric($f : x \to \mathbb{R}$)	
R_{F}	::=	FIFO	*Final rule*
	|	LIFO	
	|	Random	
k	::=	...	*Integer*

Rules and their composition





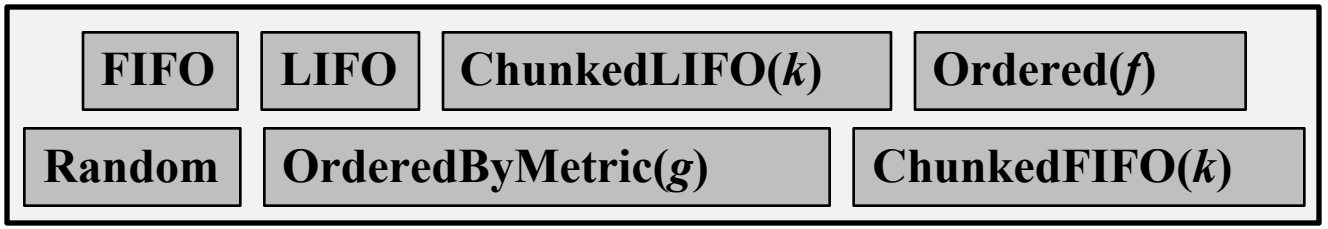
Rules and their composition

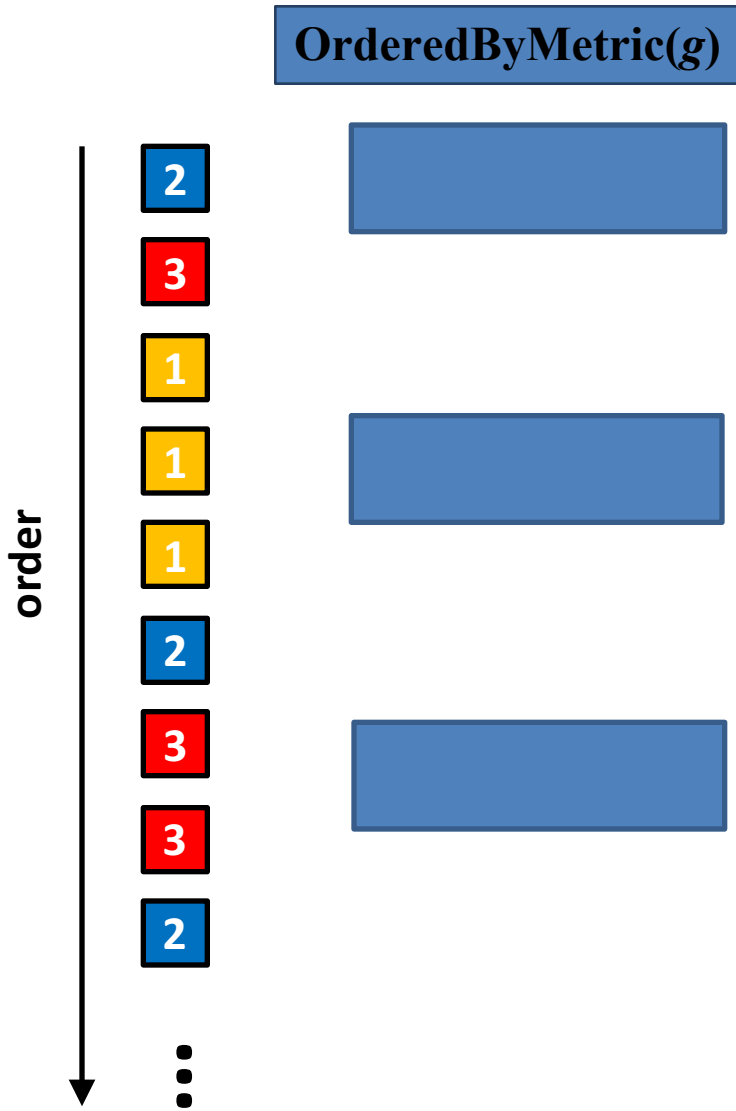


OrderedByMetric(*g*)

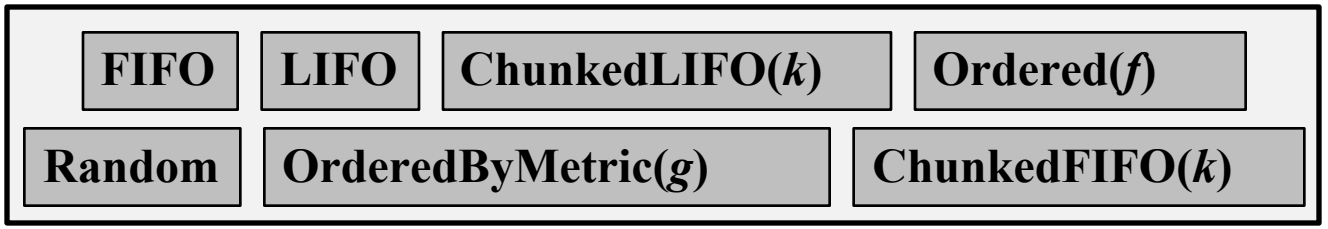
order

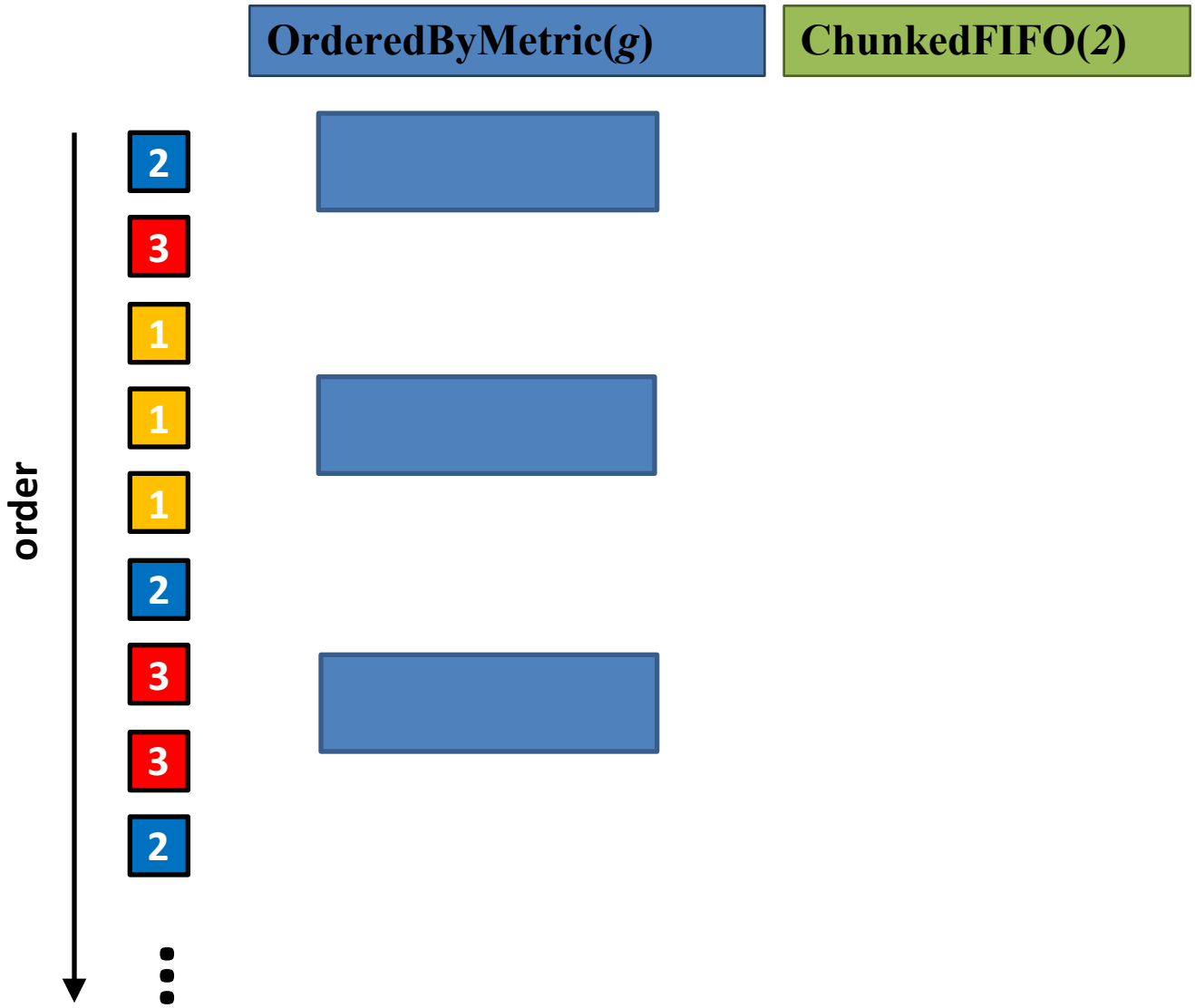
Rules and their composition



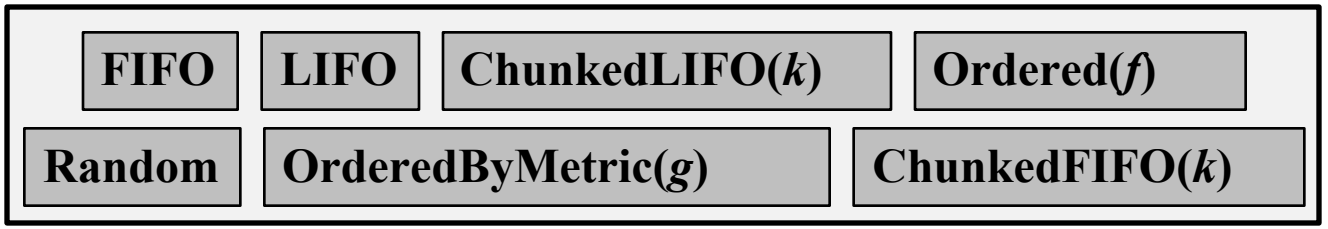


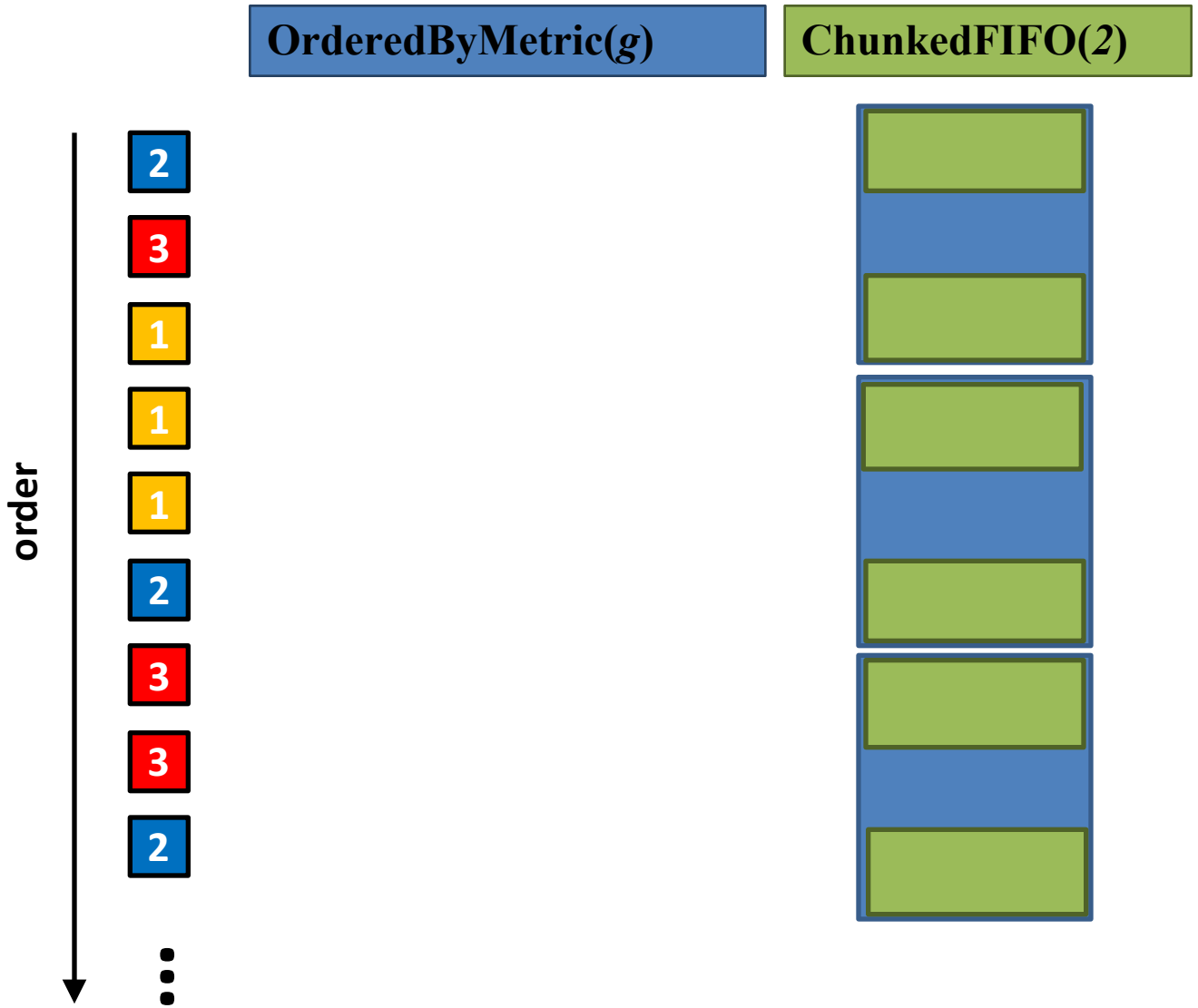
Rules and their composition



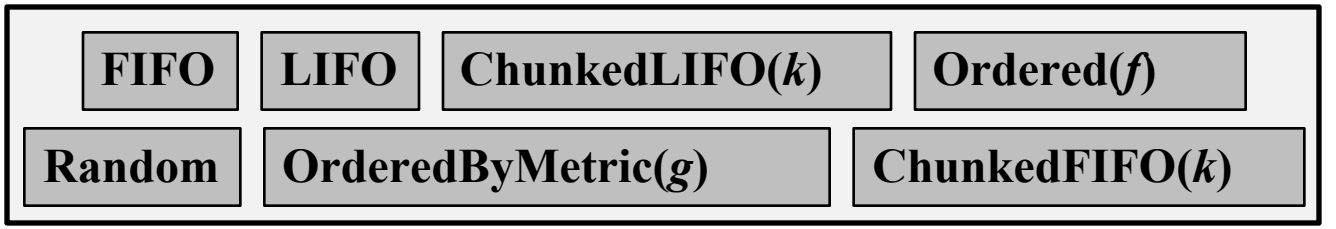


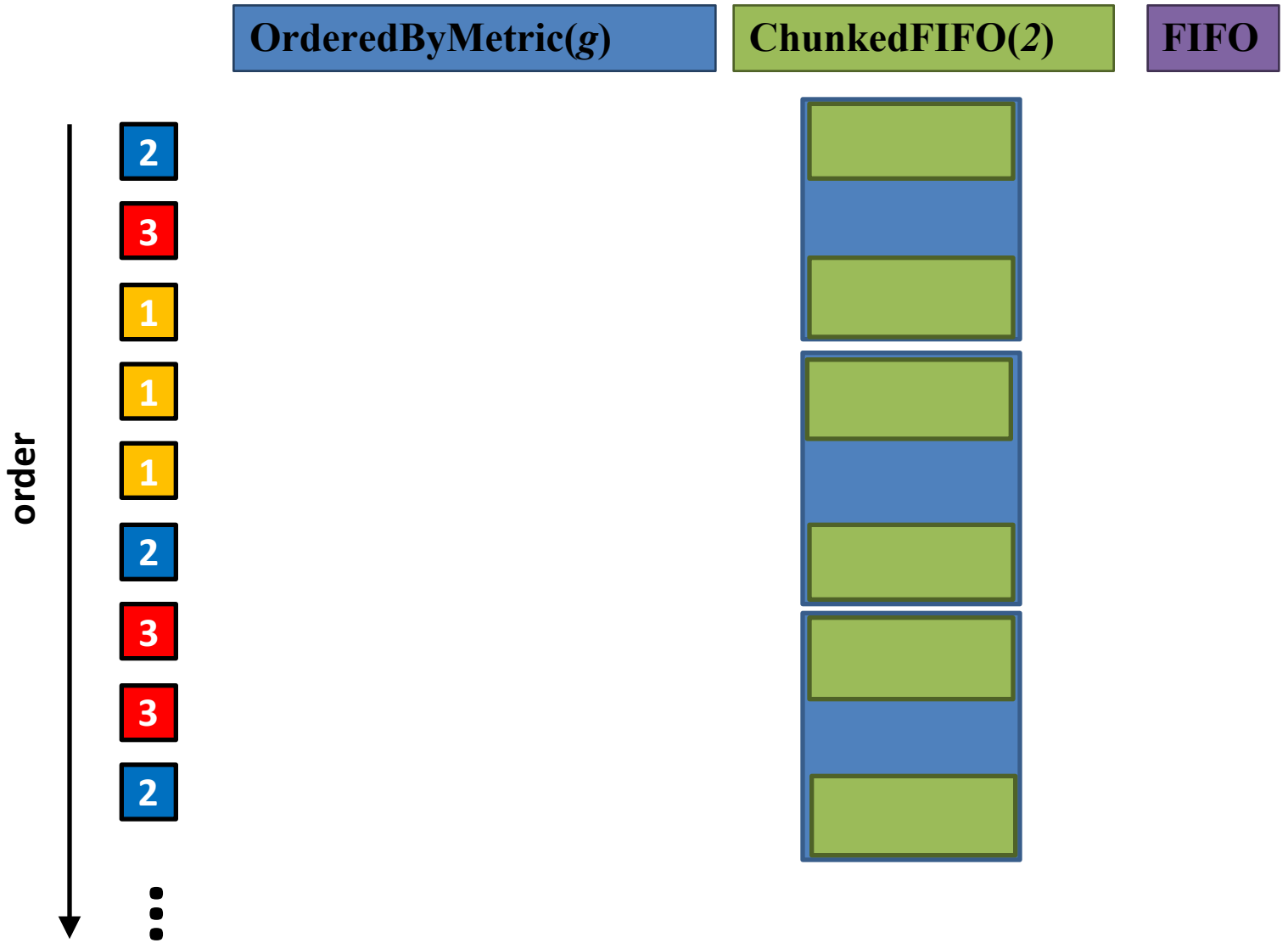
Rules and their composition





Rules and their composition

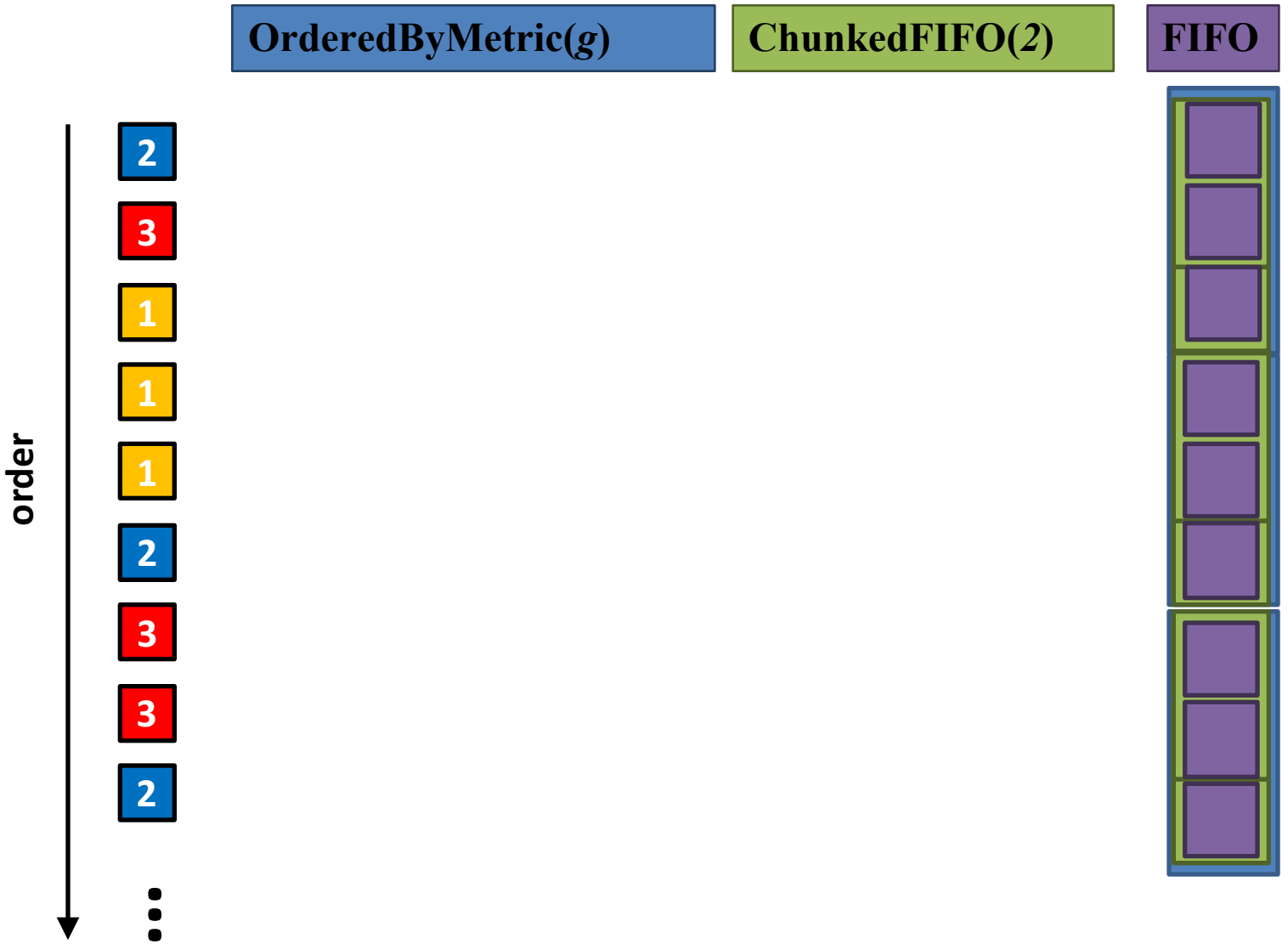




Rules and their composition

FIFO | LIFO | ChunkedLIFO(*k*) | Ordered(*f*)

Random | OrderedByMetric(*g*) | ChunkedFIFO(*k*)



Application-specific Policies

App	Order	Scheduling Policy	
PFP	FIFO	**FIFO**	[Goldberg88]
PFP	HL	**OrderedByMetric(**λ*n. -n.height*) **FIFO**	[Cherkassy95]

Application-specific Policies

App	Order	Scheduling Policy	
PFP	FIFO	**FIFO**	[Goldberg88]
PFP	HL	**OrderedByMetric(**$n.\ -n.height$**) FIFO**	[Cherkassy95]
SSSP	D-stepping	**OrderedByMetric(**$n.\ n.w / D + \ldots$**) FIFO**	[Meyer98]
SSSP	Dijkstra	**Ordered(**$a,b.\ a.w < b.w$**)**	[Dijkstra59]

Application-specific Policies

App	Order	Scheduling Policy	
PFP	FIFO	**FIFO**	[Goldberg88]
PFP	HL	**OrderedByMetric**(*n. -n.height*) **FIFO**	[Cherkassy95]
SSSP	D-stepping	**OrderedByMetric**(n. *n.w* / D + …) **FIFO**	[Meyer98]
SSSP	Dijkstra	**Ordered**(*a,b. a.w* *b.w*)	[Dijkstra59]
DMR	Local stack	**ChunkedFIFO**(*k*) Local: **LIFO**	[Kulkarni08]

Application-specific Policies

App	Order	Scheduling Policy	
PFP	FIFO	**FIFO**	[Goldberg88]
PFP	HL	**OrderedByMetric**(*n. -n.height*) **FIFO**	[Cherkassy95]
SSSP	D-stepping	**OrderedByMetric**(n. *n.w* / D + …) **FIFO**	[Meyer98]
SSSP	Dijkstra	**Ordered**(*a,b. a.w* *b.w*)	[Dijkstra59]
DMR	Local stack	**ChunkedFIFO**(*k*) Local: **LIFO**	[Kulkarni08]
DT	BRIO	**OrderedByMetric**(*p. p.rnd*) **ChunkedFIFO**(*k*)	[Amenta03]

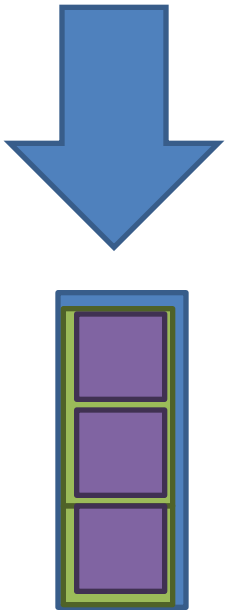
Application-specific Policies

App	Order	Scheduling Policy	
PFP	FIFO	**FIFO**	[Goldberg88]
PFP	HL	**OrderedByMetric**(*n. -n.height*) **FIFO**	[Cherkassy95]
SSSP	D-stepping	**OrderedByMetric**(n. *n.w* / D + …) **FIFO**	[Meyer98]
SSSP	Dijkstra	**Ordered**(*a,b. a.w* *b.w*)	[Dijkstra59]
DMR	Local stack	**ChunkedFIFO**(*k*) Local: **LIFO**	[Kulkarni08]
DT	BRIO	**OrderedByMetric**(*p. p.rnd*) **ChunkedFIFO**(*k*)	[Amenta03]
PTA	Split		[Nielson99]

Outline

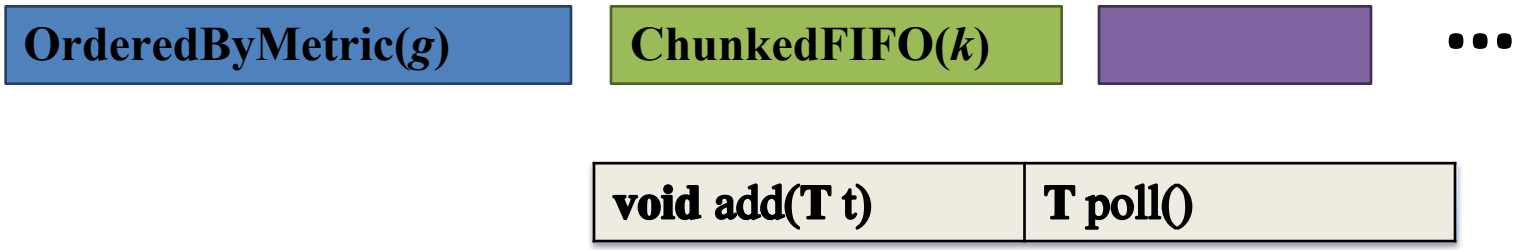
1. Language
2. **Synthesis**
3. Results

P	::=	Global:D Local:D	*Specification*
D	::=	$R_{\mathrm{NF}}*$ $R_{\mathrm{F}}?$	*Ordering*
R_{NF}	::=	ChunkedFIFO(k)	*Non-final rule*
	|	ChunkedLIFO(k)	
	|	Ordered($f : x, y \to \mathrm{bool}$)	
	|	OrderedByMetric($f : x \to \mathbb{R}$)	
R_{F}	::=	FIFO	*Final rule*
	|	LIFO	
	|	Random	
k	::=	...	*Integer*

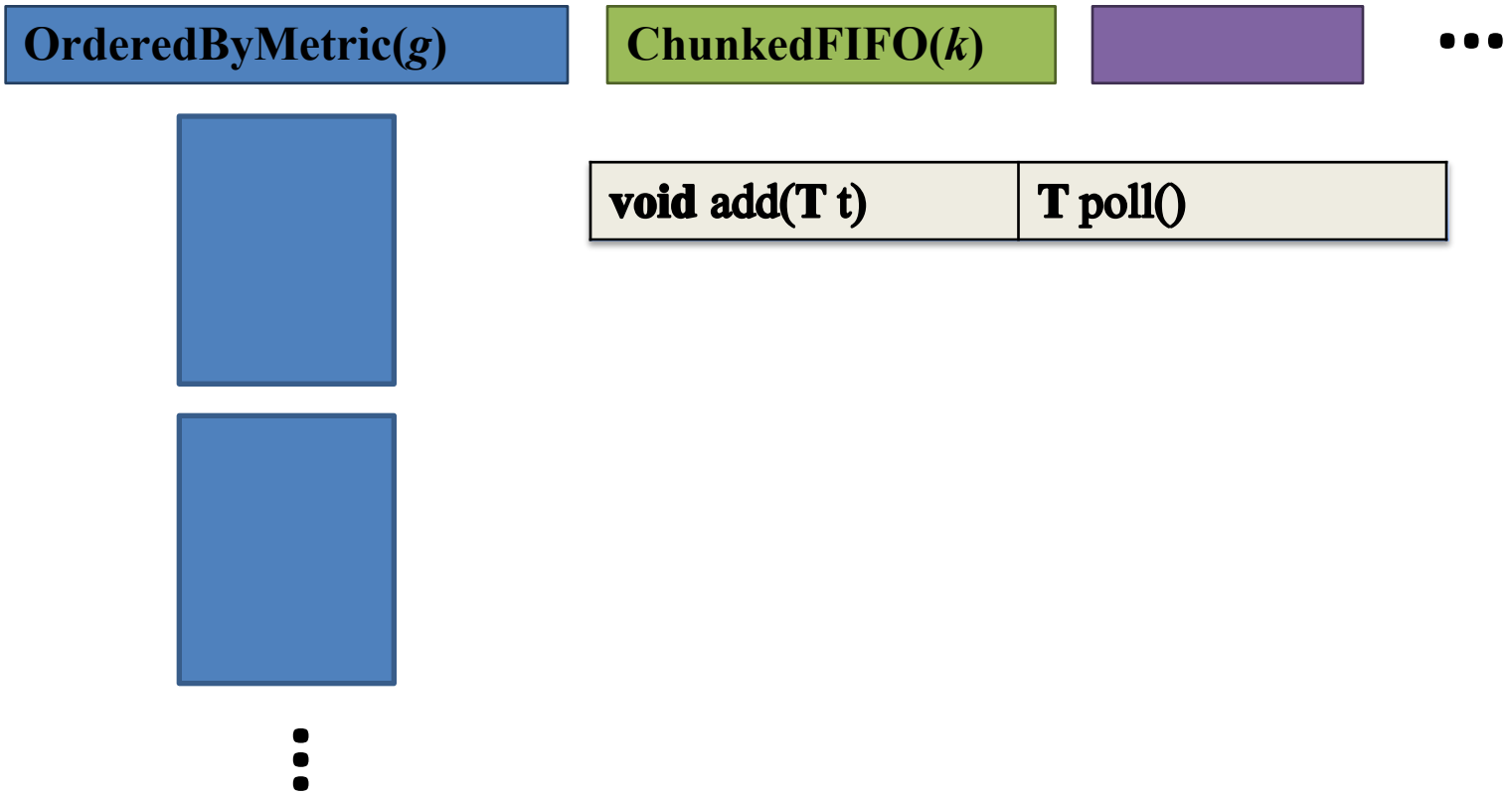


Synthesis of Serial Schedulers

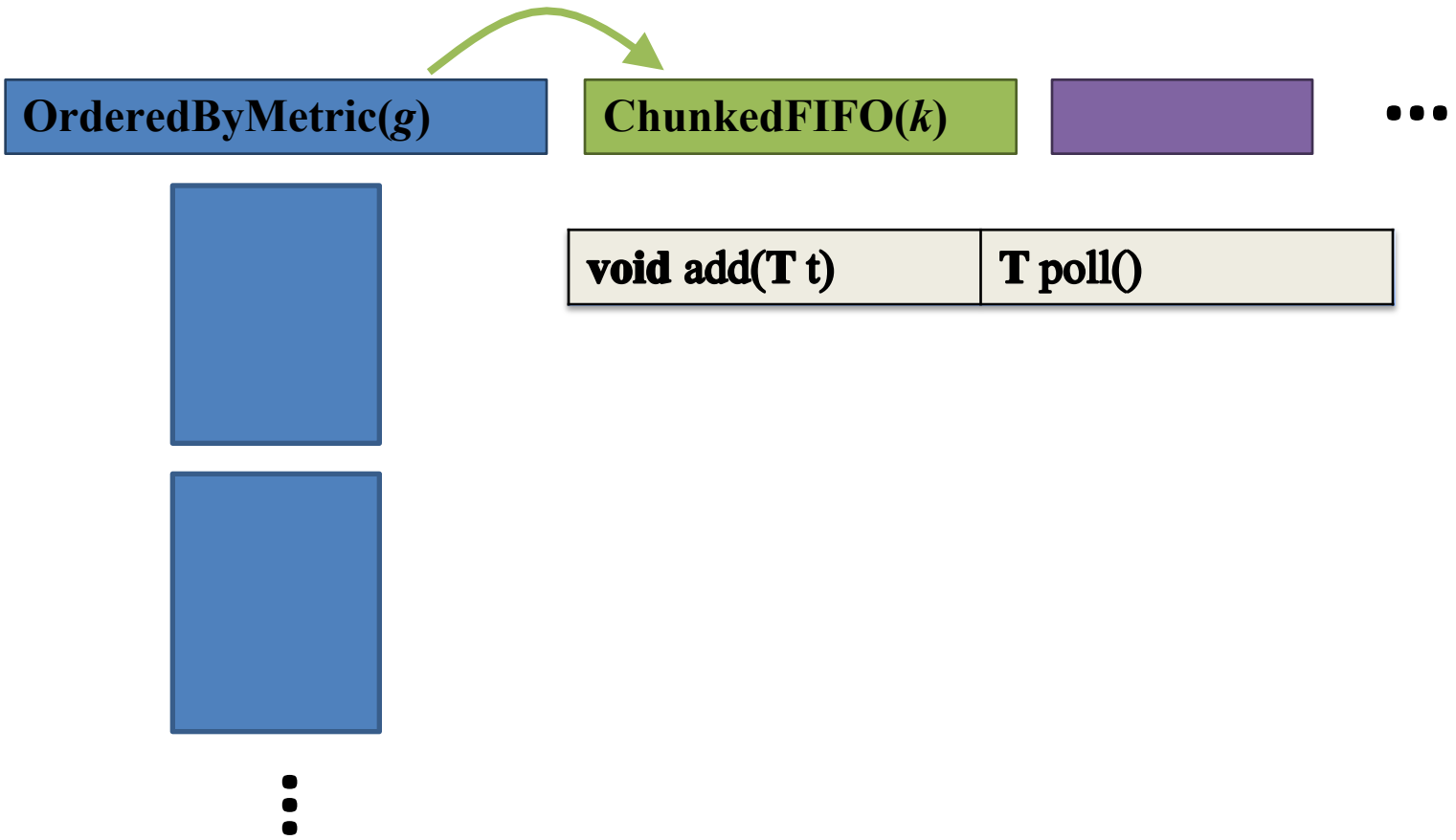
Synthesis of Serial Schedulers



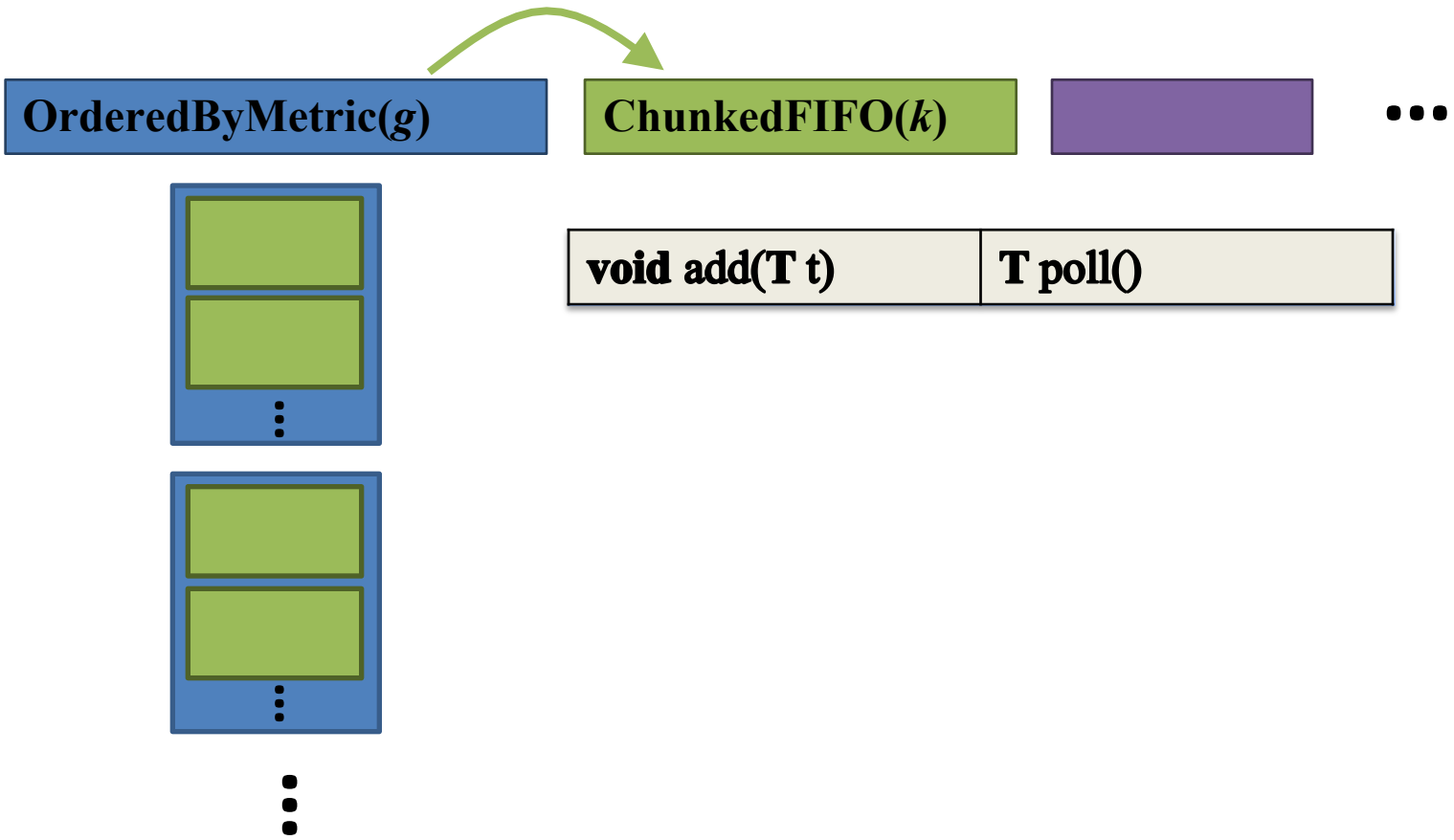
Synthesis of Serial Schedulers



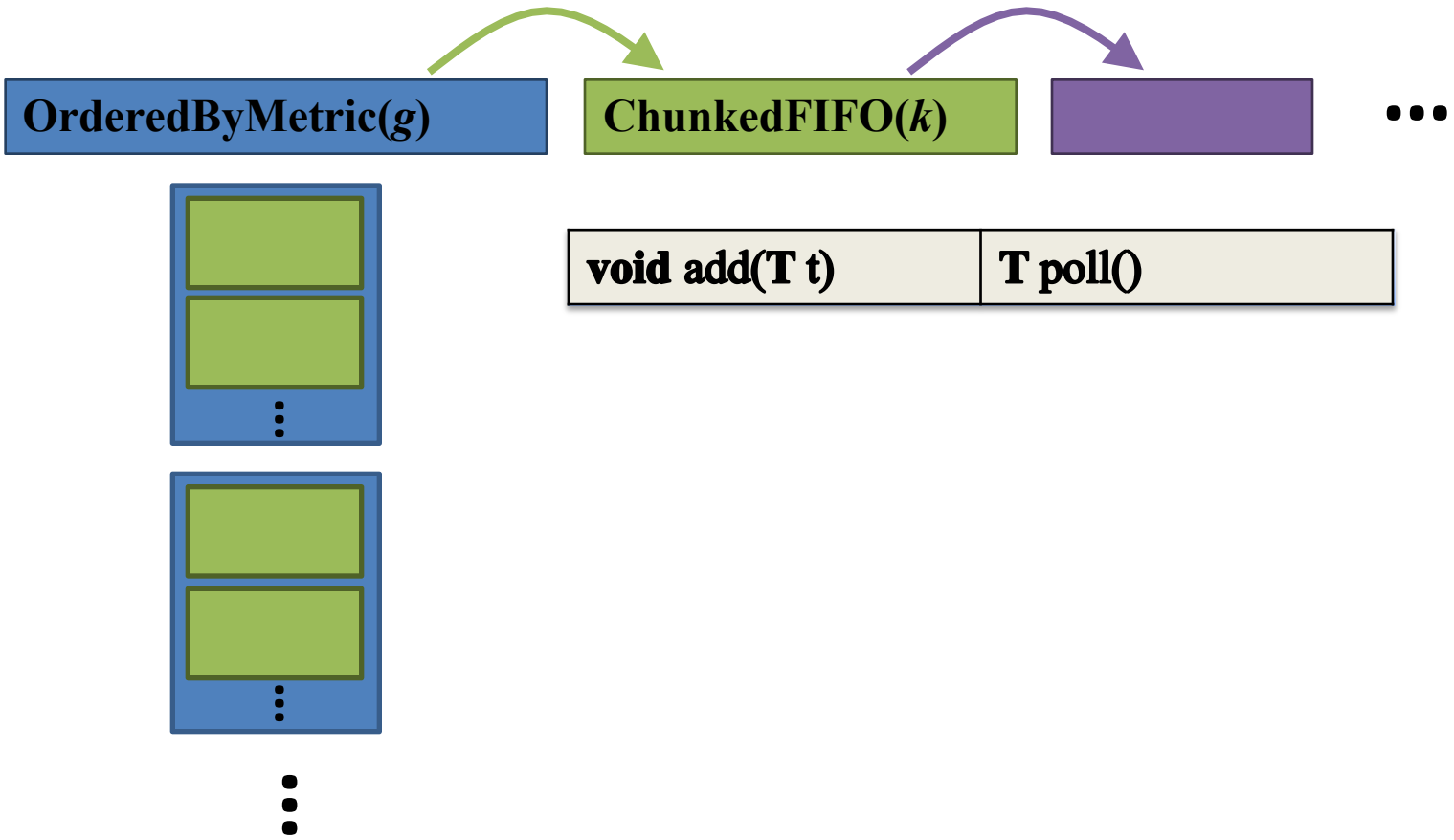
Synthesis of Serial Schedulers



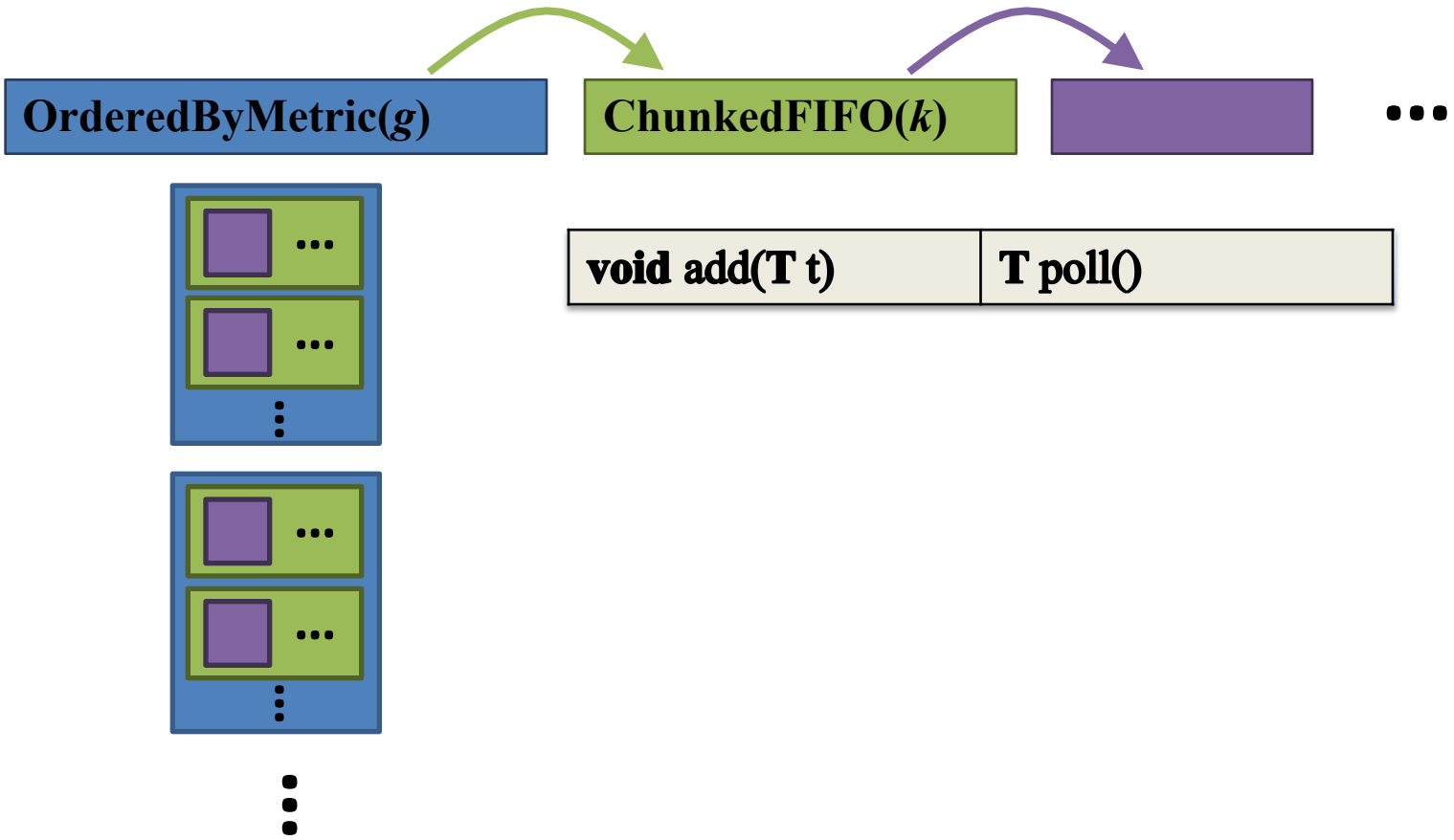
Synthesis of Serial Schedulers



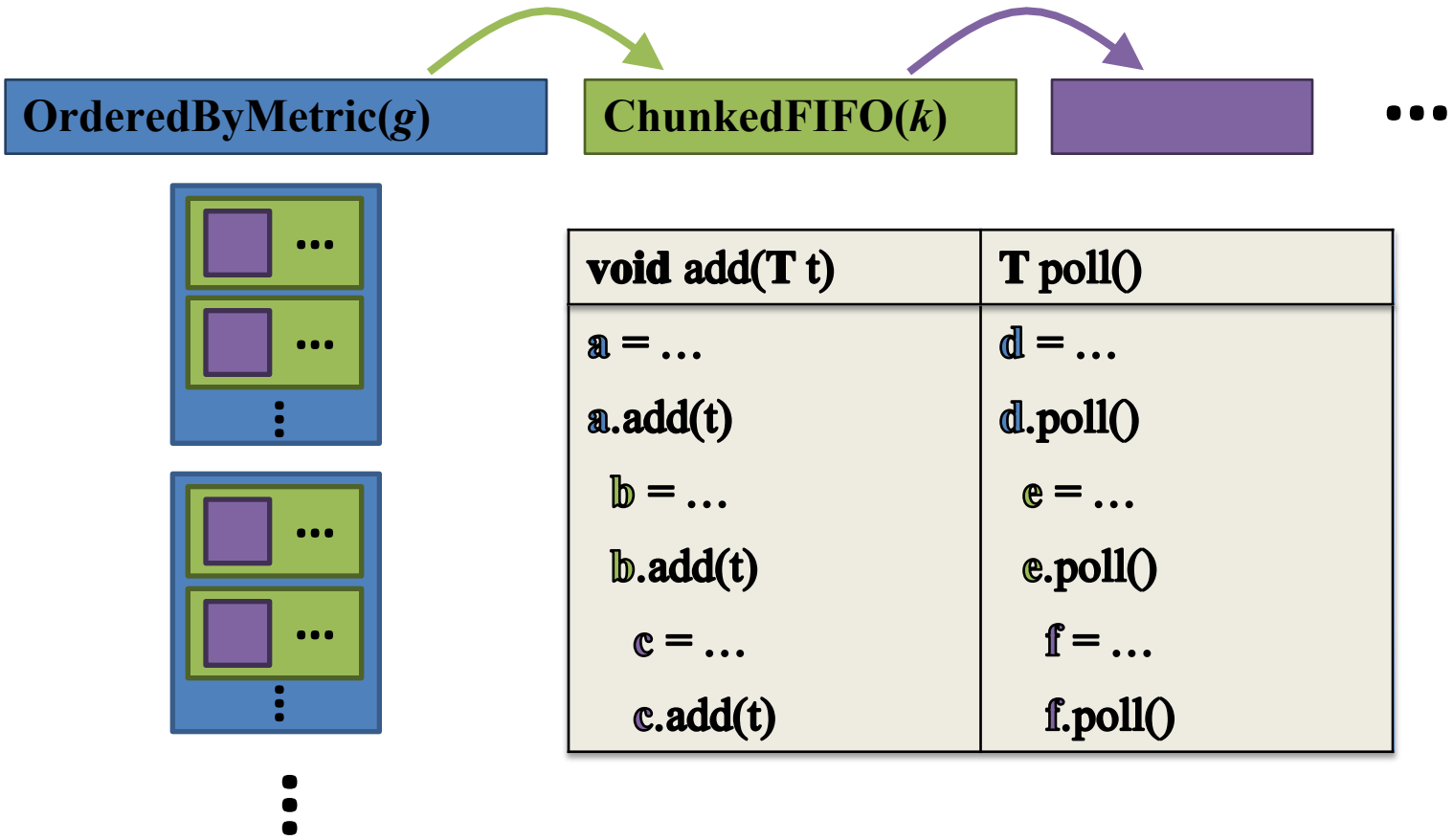
Synthesis of Serial Schedulers



Synthesis of Serial Schedulers

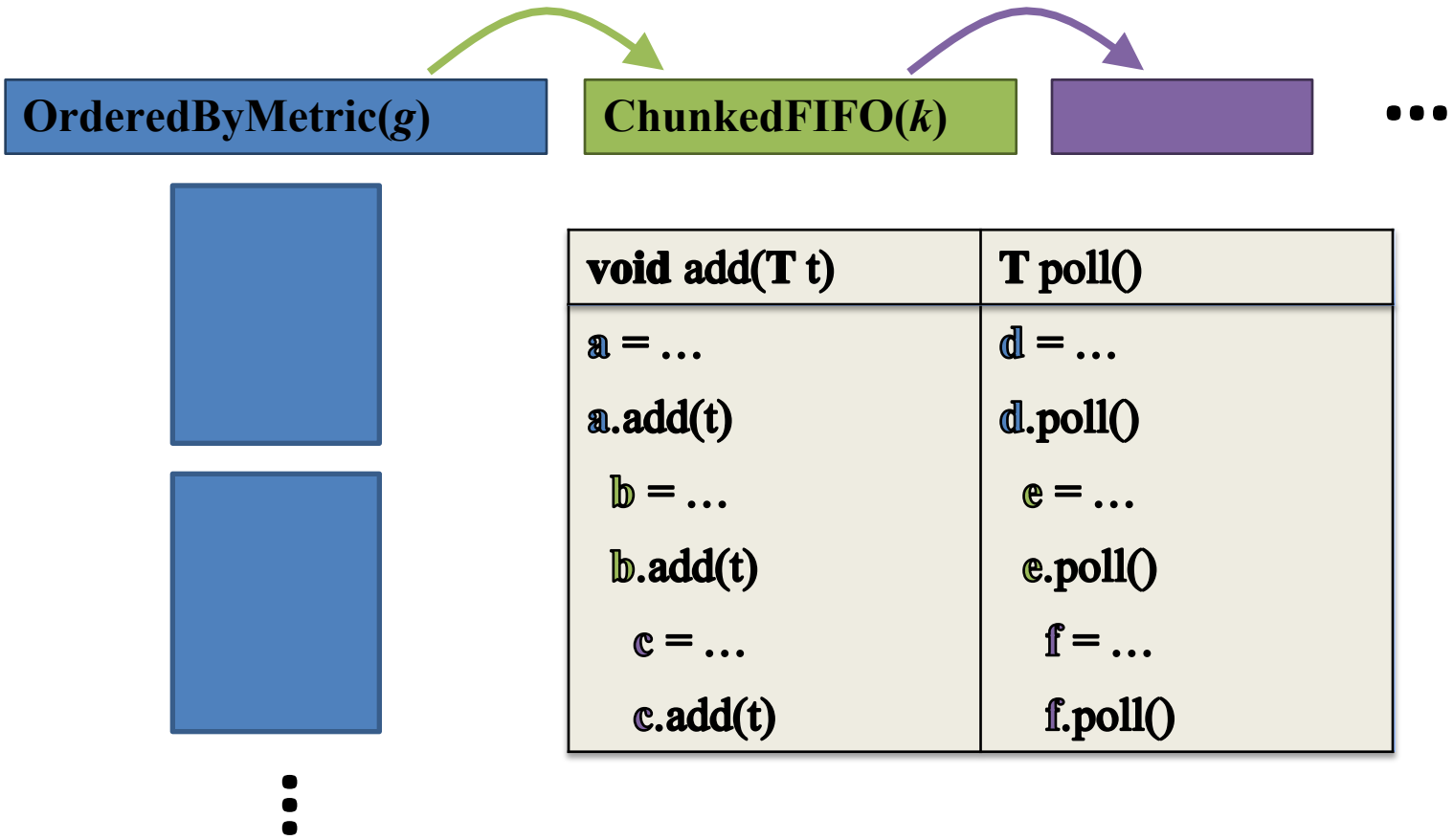


Synthesis of Serial Schedulers



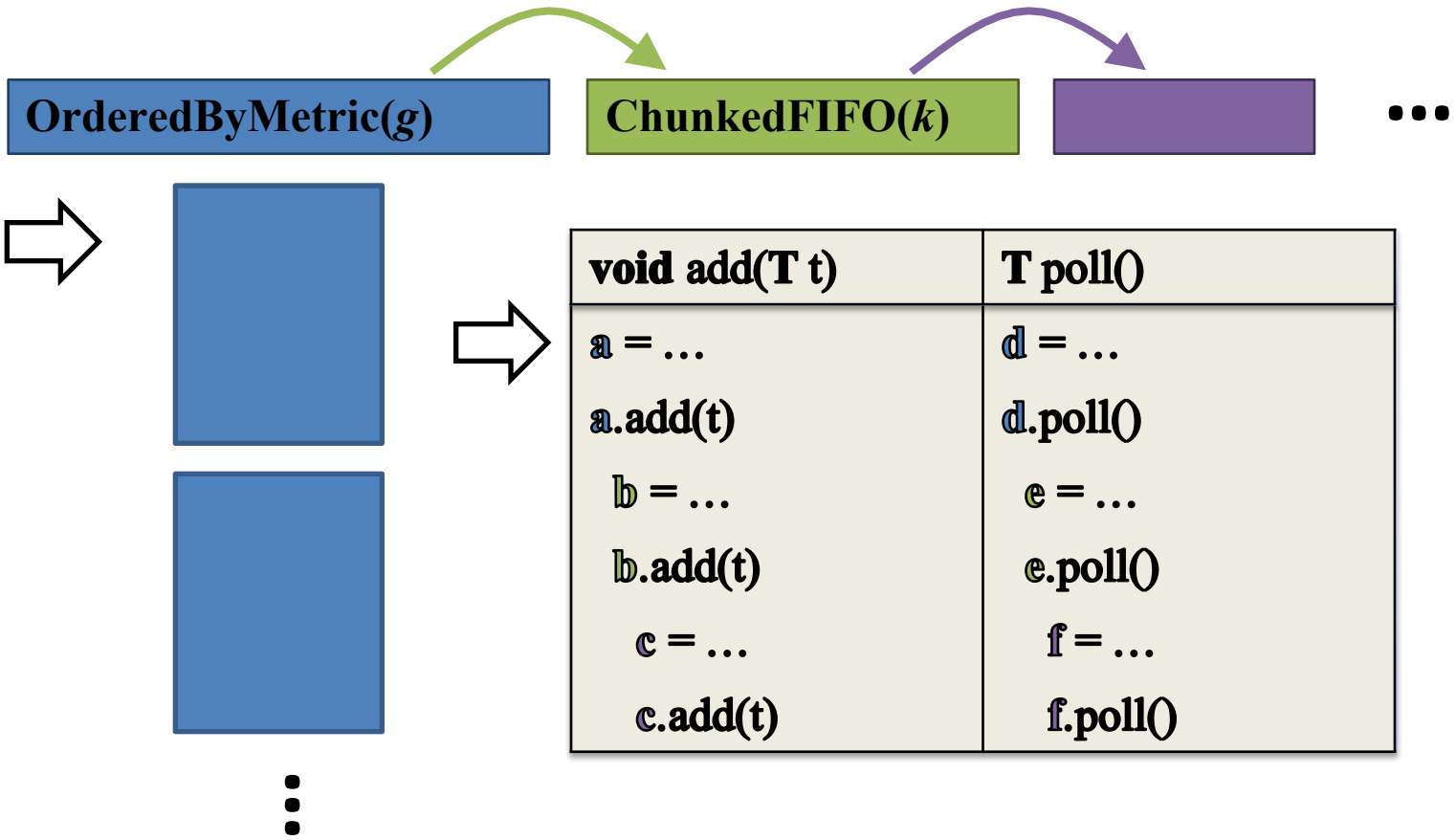
void add(T t)	T poll()
a = …	d = …
a.add(t)	d.poll()
b = …	e = …
b.add(t)	e.poll()
c = …	f = …
c.add(t)	f.poll()

Synthesis of Serial Schedulers

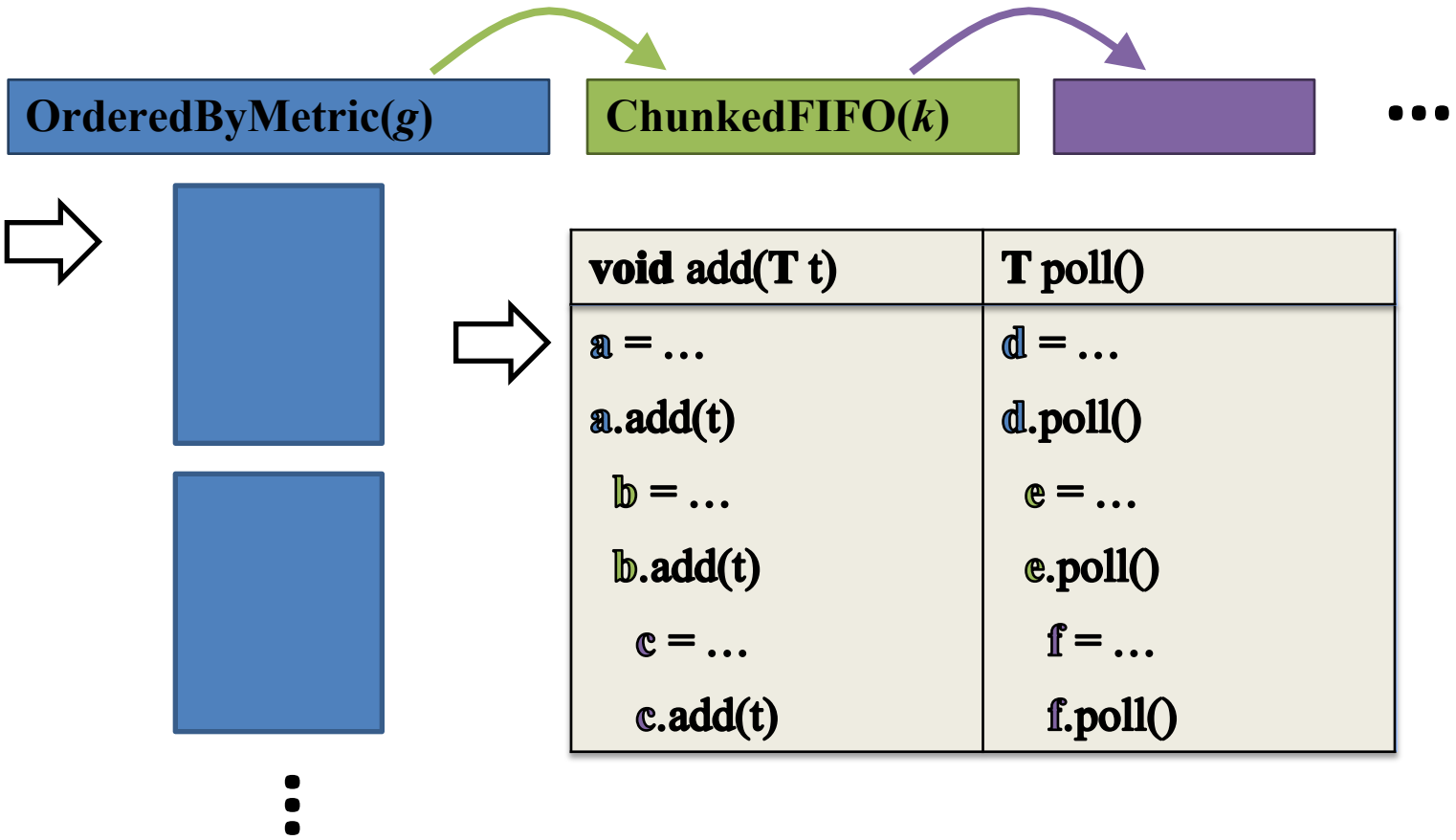


void add(T t)	T poll()
a = …	d = …
a.add(t)	d.poll()
b = …	e = …
b.add(t)	e.poll()
c = …	f = …
c.add(t)	f.poll()

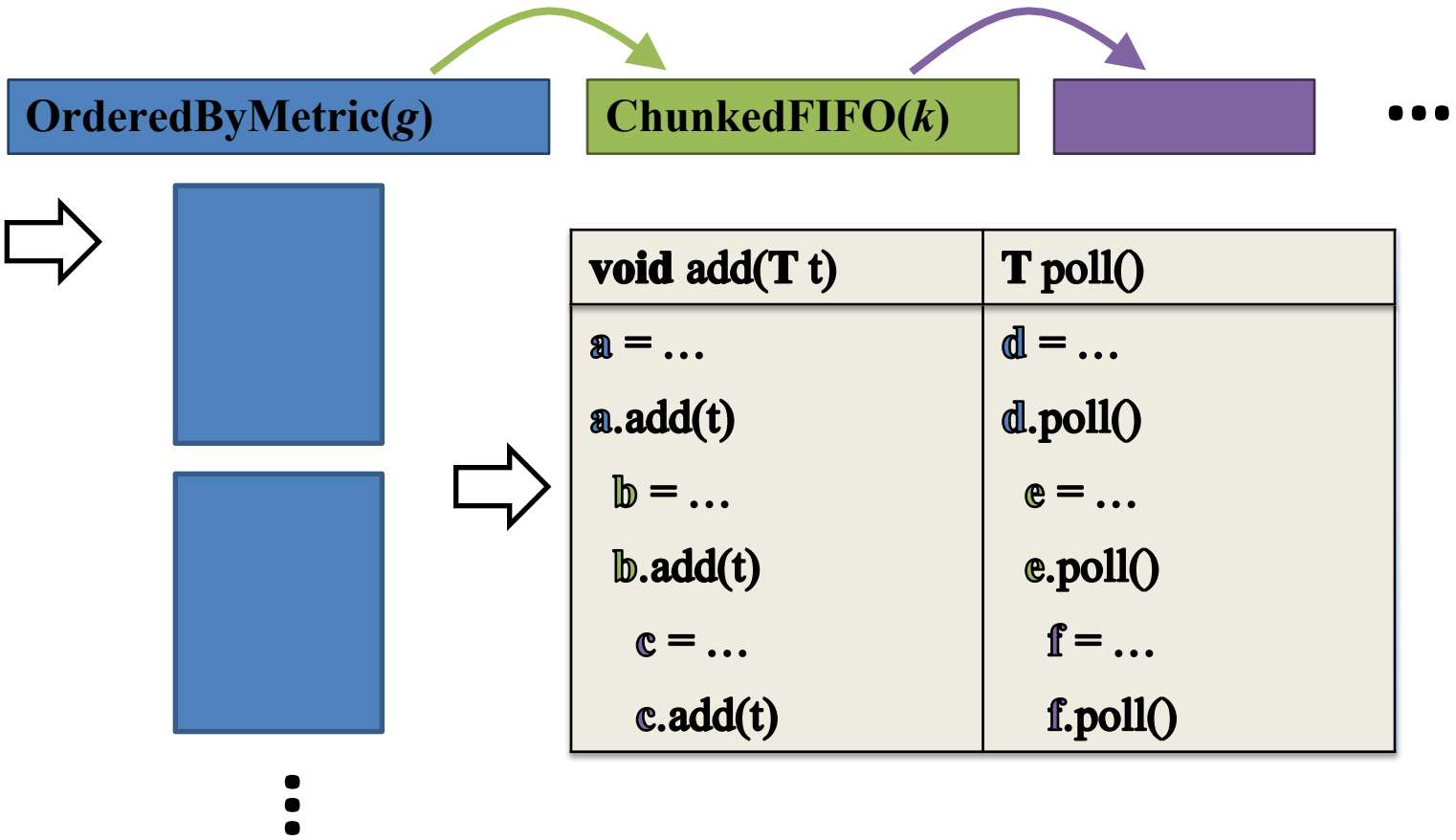
Synthesis of Serial Schedulers



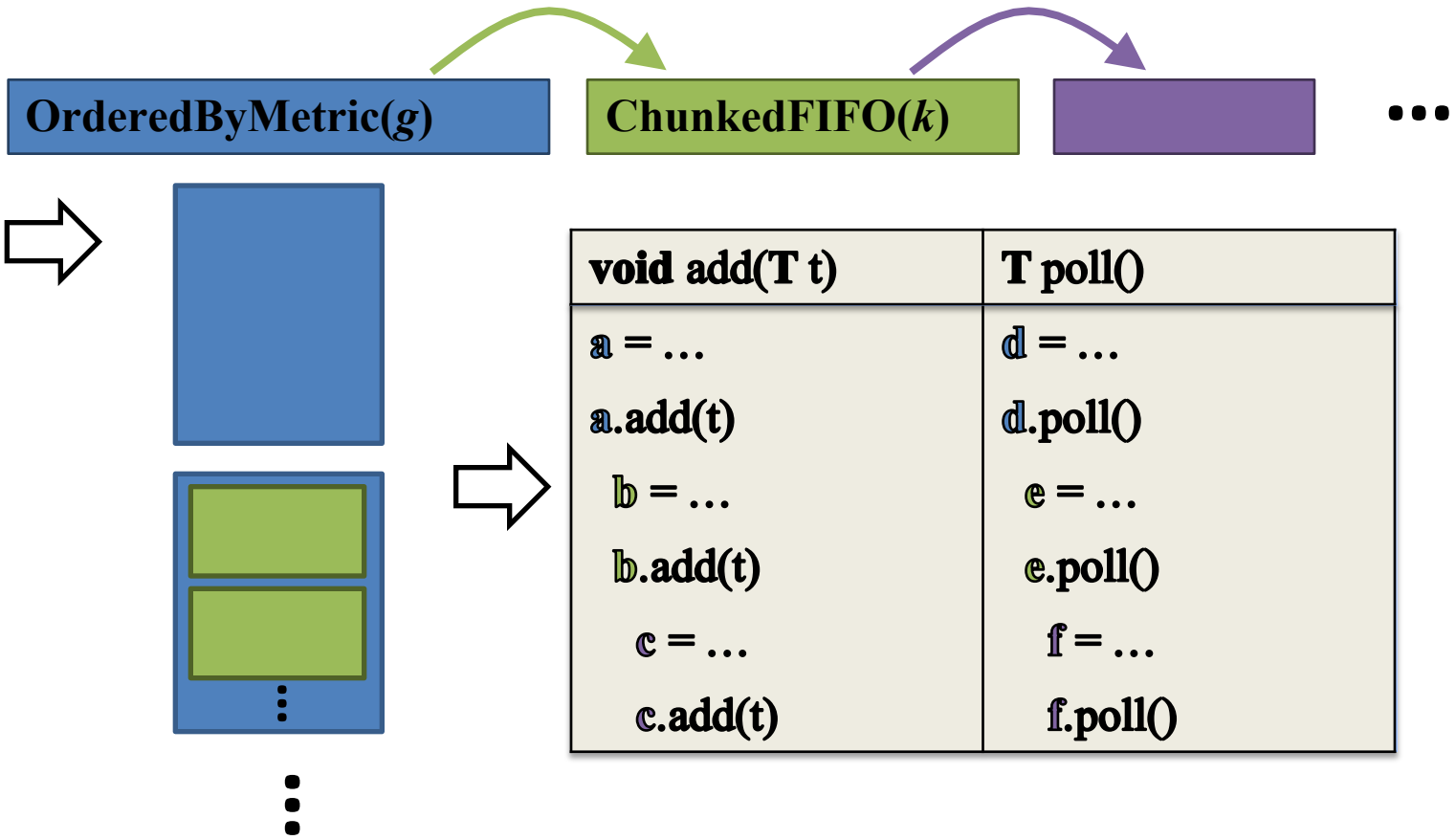
Synthesis of Serial Schedulers



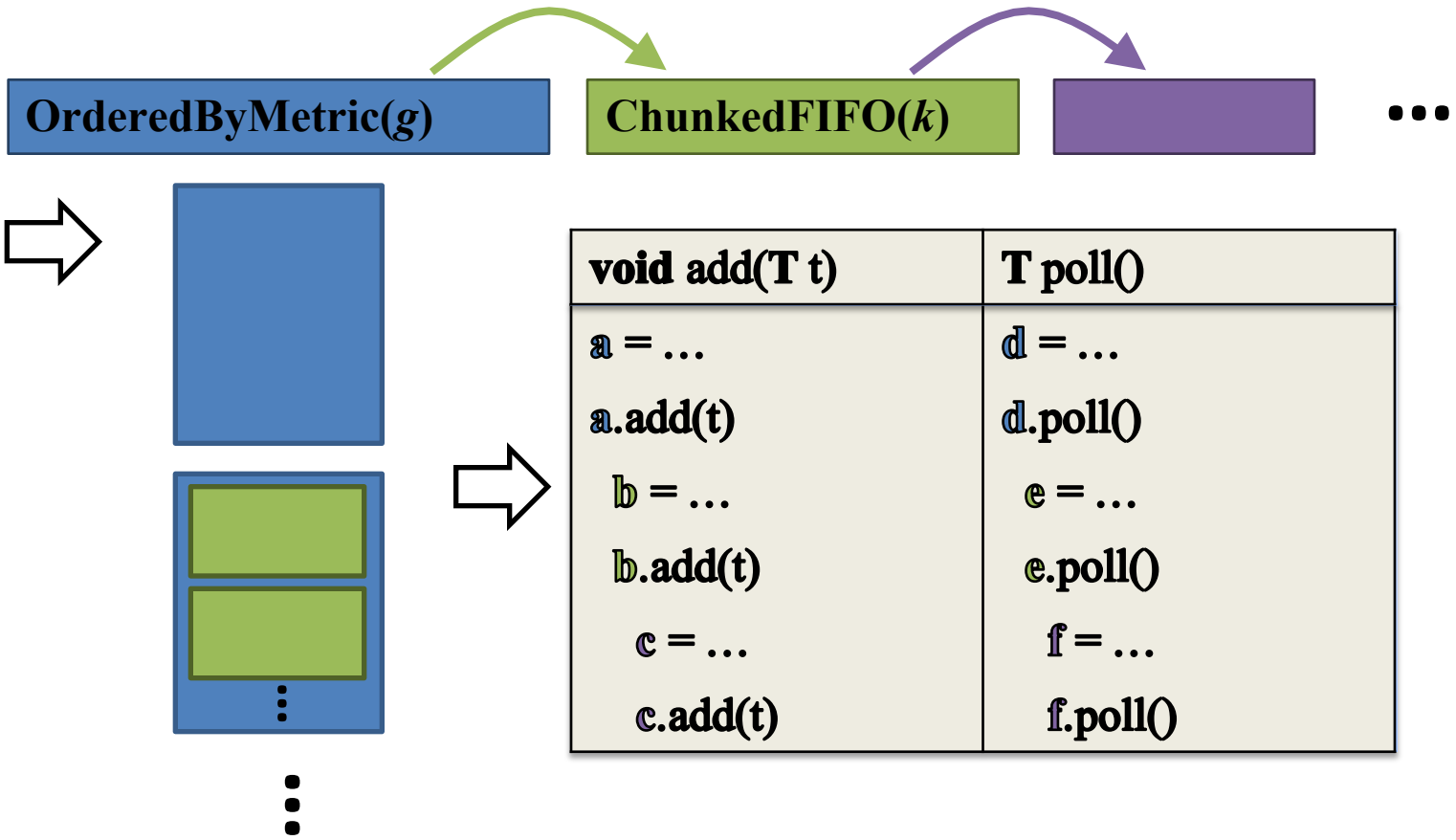
Synthesis of Serial Schedulers



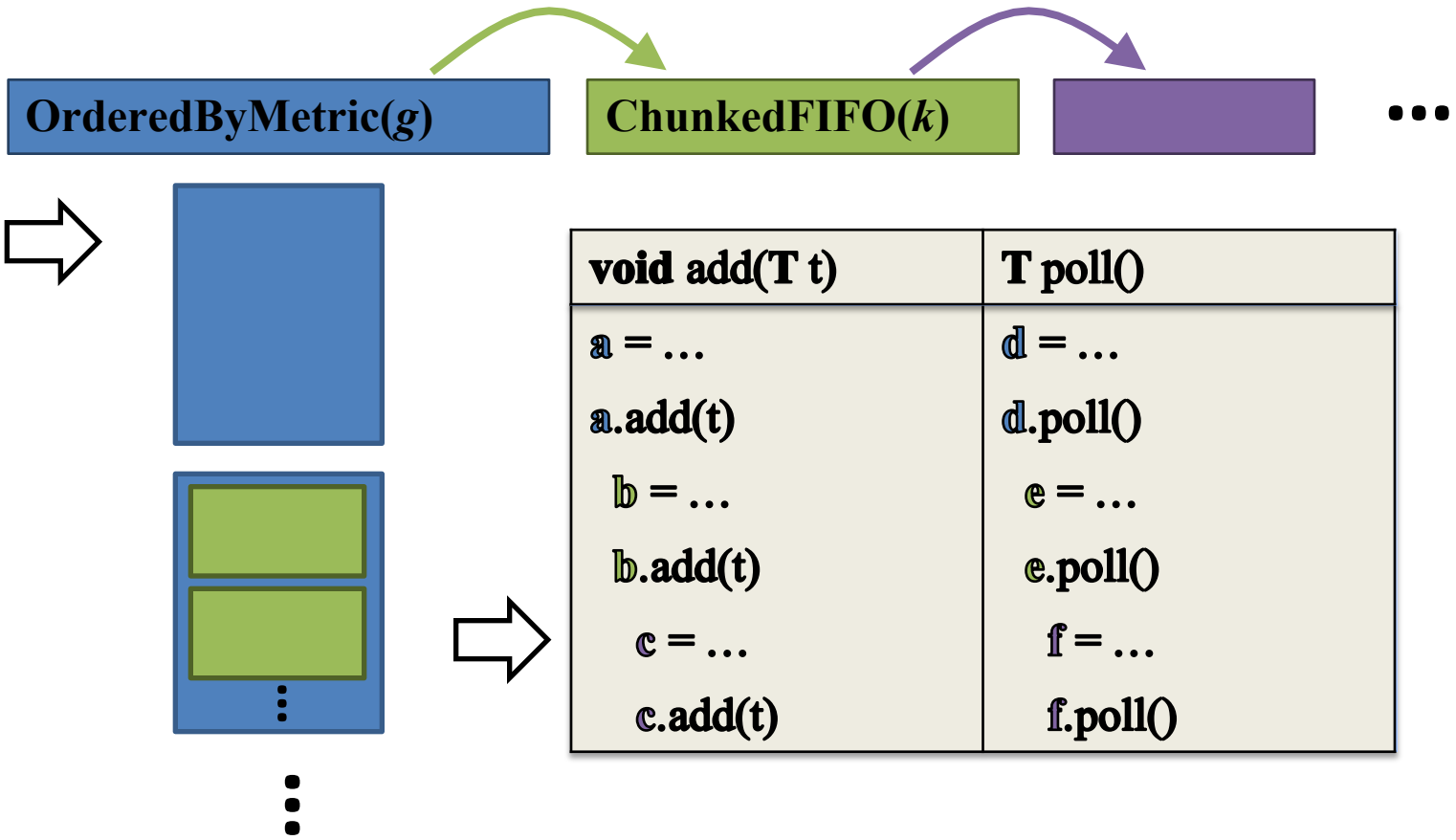
Synthesis of Serial Schedulers



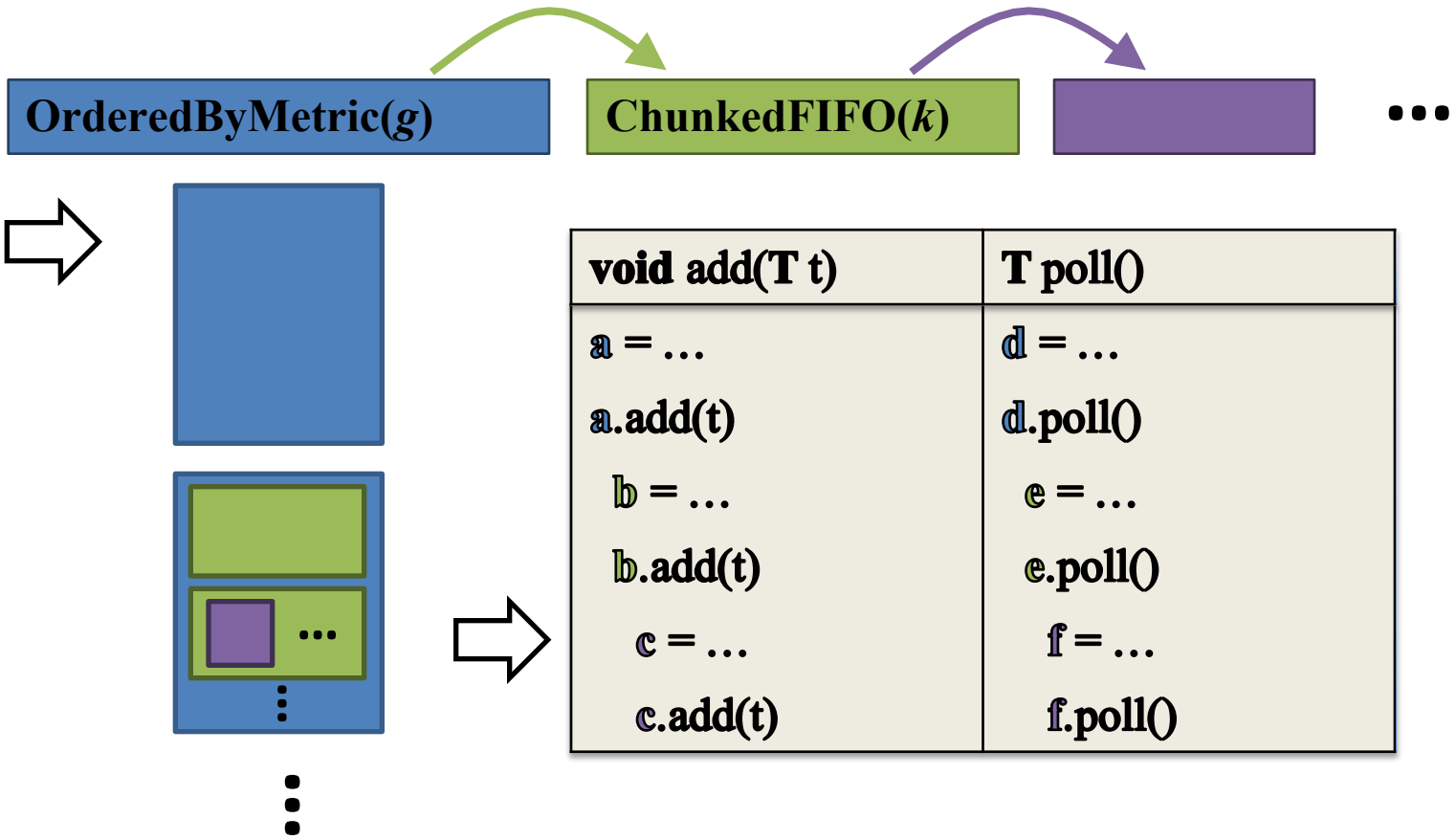
Synthesis of Serial Schedulers



Synthesis of Serial Schedulers



Synthesis of Serial Schedulers

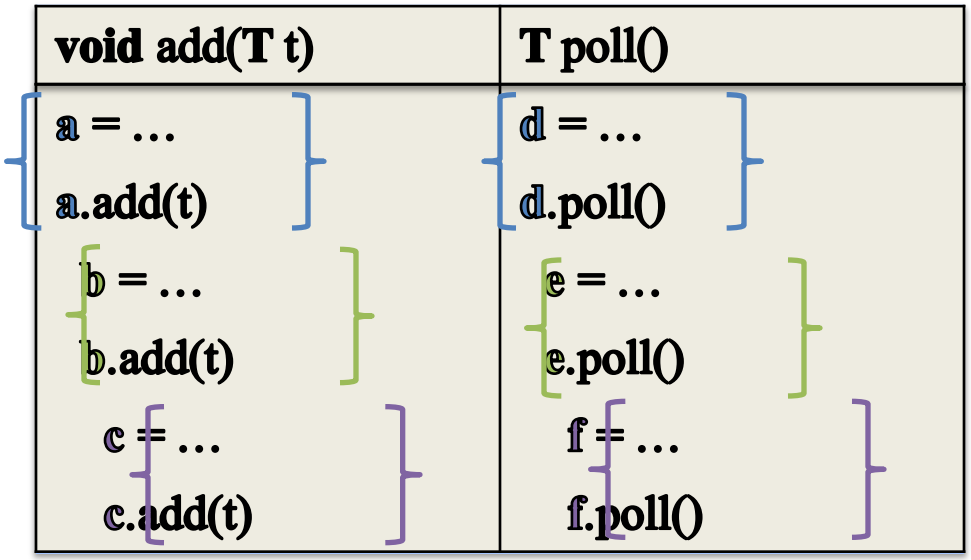


Synthesis of Concurrent Schedulers

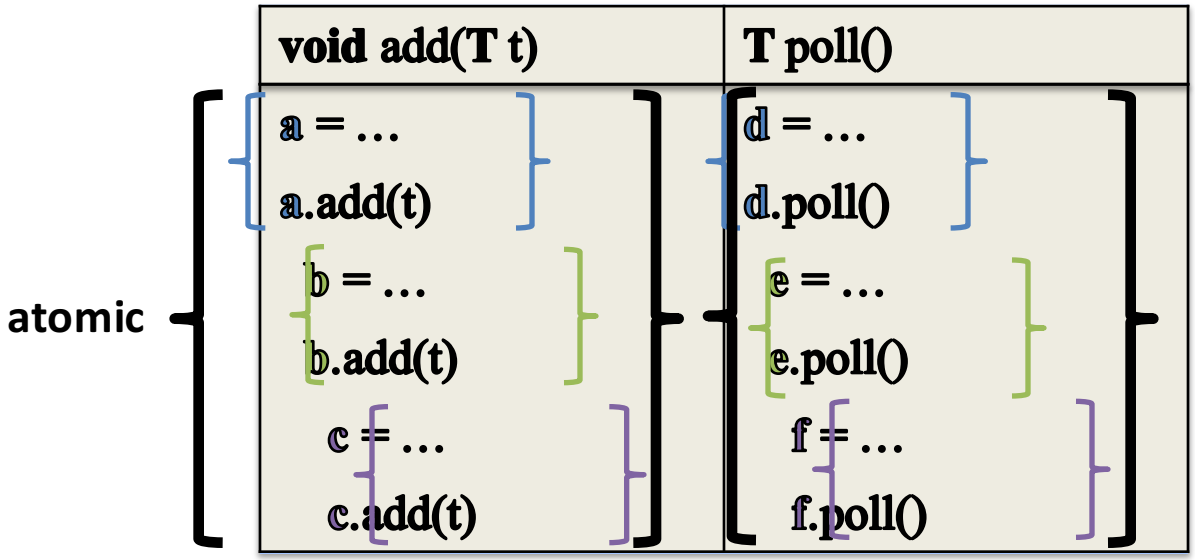
void add(T t)	T poll()
a = …	d = …
a.add(t)	d.poll()
b = …	e = …
b.add(t)	e.poll()
c = …	f = …
c.add(t)	f.poll()

Synthesis of Concurrent Schedulers

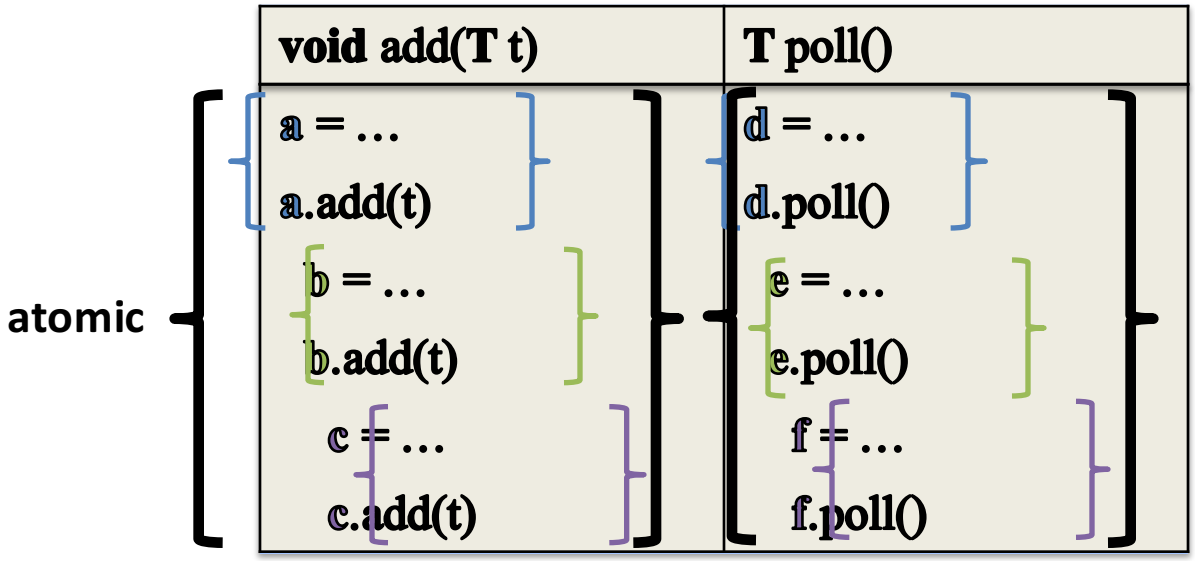
atomic



Synthesis of Concurrent Schedulers



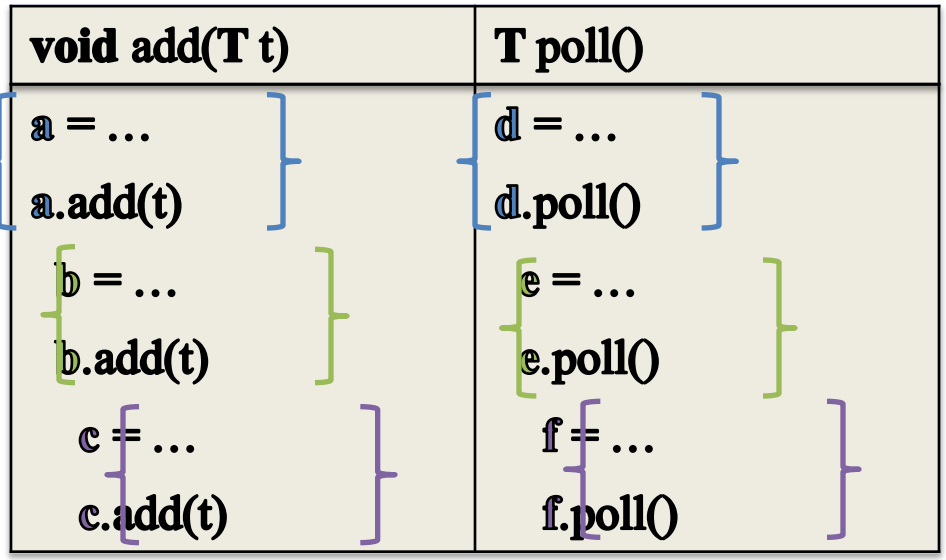
Synthesis of Concurrent Schedulers



- Key property
  - Scheduling orders are heuristics
  - **Scheduler is valid as long as we eventually retrieve every item added (in some order)**

Synthesis of Concurrent Schedulers

atomic



- Key property
  - Scheduling orders are heuristics
  - **Scheduler is valid as long as we eventually retrieve every item added (in some order)**

Synthesis of Concurrent Schedulers

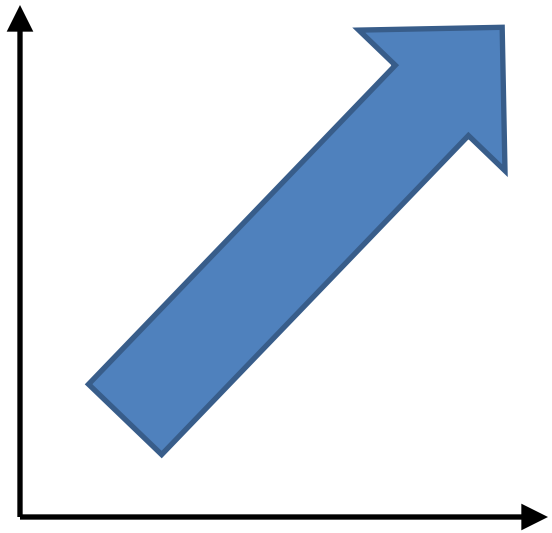
atomic

void add(T t)	T poll()	T poll-s()
a = … a.add(t)	d = … d.poll()	d = … d.poll()
b = … b.add(t)	e = … e.poll()	e = … e.poll()
c = … c.add(t)	f = … f.poll()	f = … f.poll()

- Key property
  - Scheduling orders are heuristics
  - **Scheduler is valid as long as we eventually retrieve every item added (in some order)**

Outline

1. Language
2. Synthesis
3. **Results**



Methodology

- Three machines
  - Nehalem: 2x4-core @ 2.93GHz (Xeon X5570)
  - Shanghai: 4x4-core @ 2.7GHz (Opteron 8384)
  - Niagara: 4x8-core @ 1.4GHz (UltraSPARC T2 Plus)
- Java
  - Sun JDK 1.6
  - 20GB heap
  - Last of three runs within same VM instance

Scheduling Policies

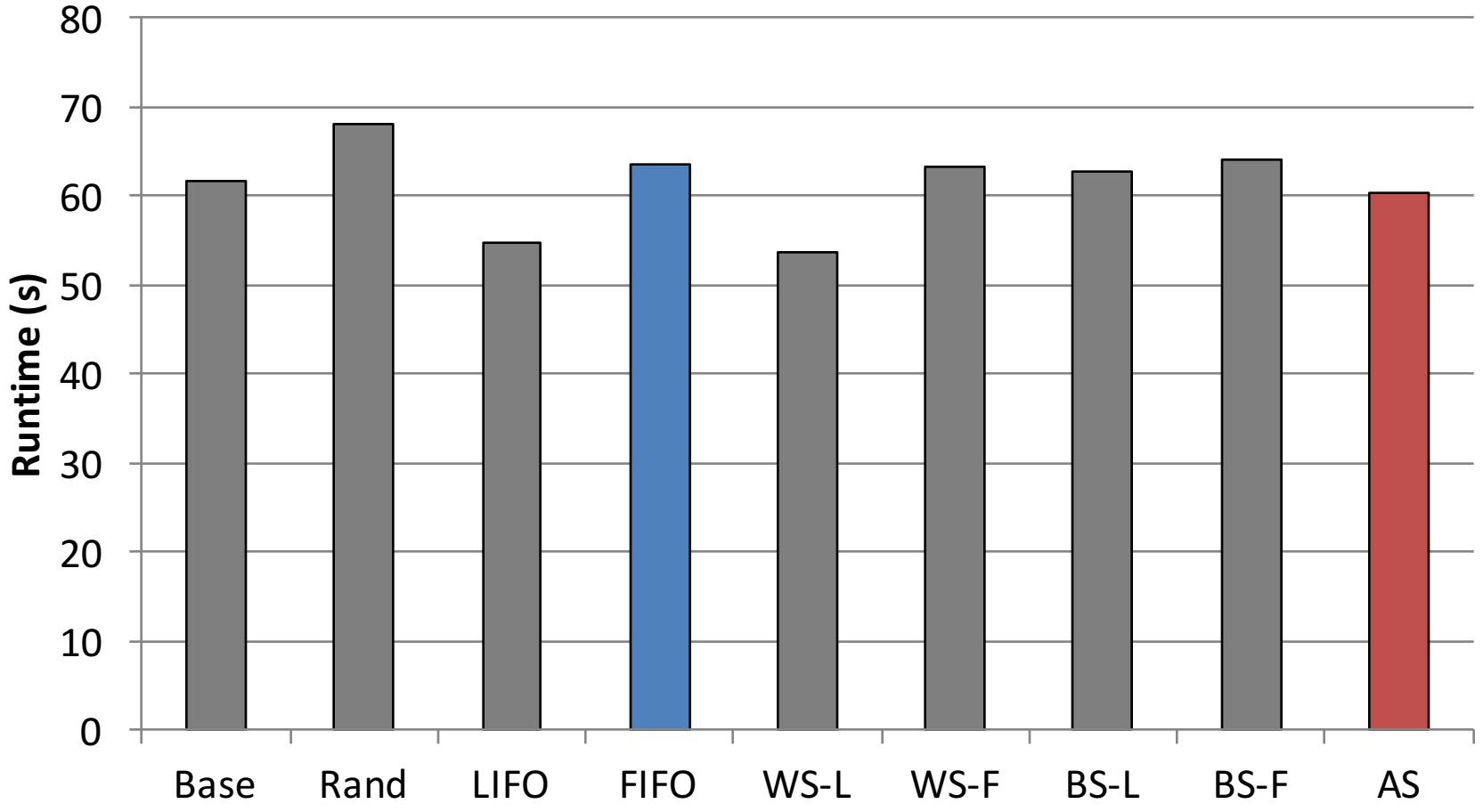
- Fixed-forms
  - **Rand**
  - **LIFO**, **FIFO**: Global queue or stack
  - **WS-L**, **WS-F**: Work-stealing with queue or stack
  - **BS-L**, **BS-F**

- Synthesized
  - **Base**: ChunkedFIFO(32)
  - **AS**: Application-specific policy

Application-specific Policies

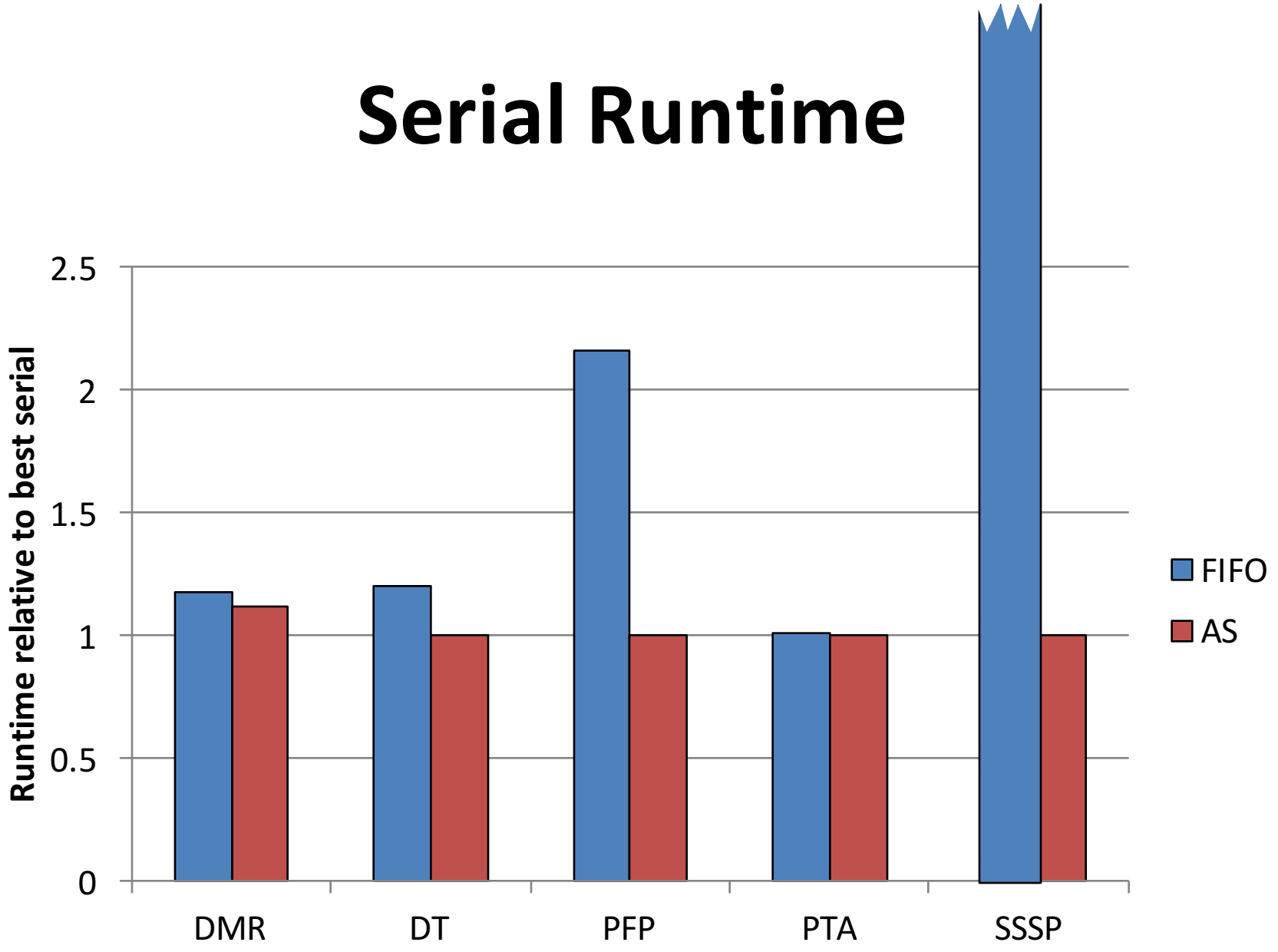
App	Order	Scheduling Policy	
PFP	FIFO	**FIFO**	[Goldberg88]
PFP	HL	**OrderedByMetric**(*n. -n.height*) **FIFO**	[Cherkassy95]
SSSP	D-stepping	**OrderedByMetric**(n. *n.w* / D + …) **FIFO**	[Meyer98]
SSSP	Dijkstra	**Ordered**(*a,b. a.w* *b.w*)	[Dijkstra59]
DMR	Local stack	**ChunkedFIFO**(*k*) Local: **LIFO**	[Kulkarni08]
DT	BRIO	**OrderedByMetric**(*p. p.rnd*) **ChunkedFIFO**(*k*)	[Amenta03]
PTA	Split	*BS-F*	[Nielson99]

Serial Runtime: DMR



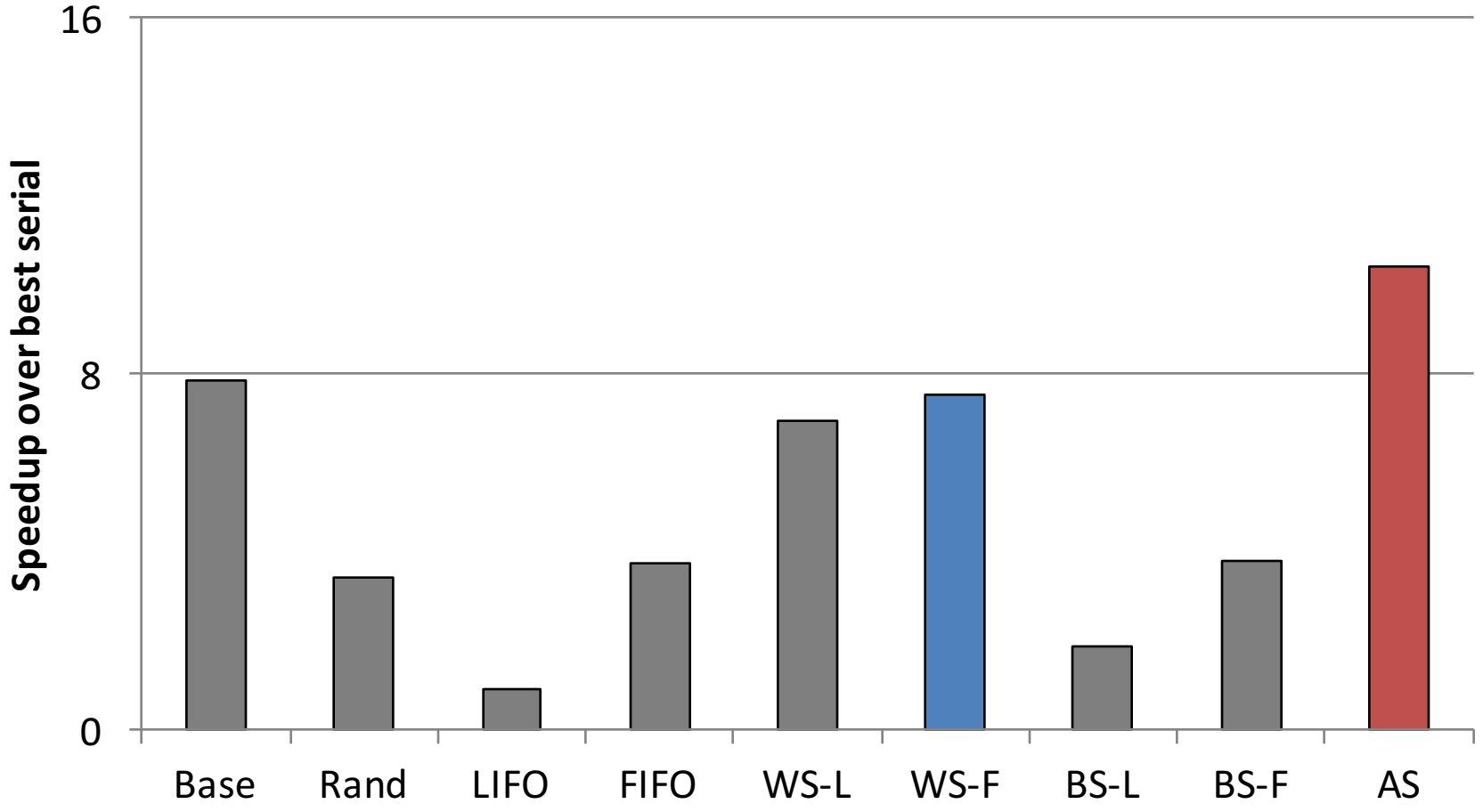
shanghai

Serial Runtime



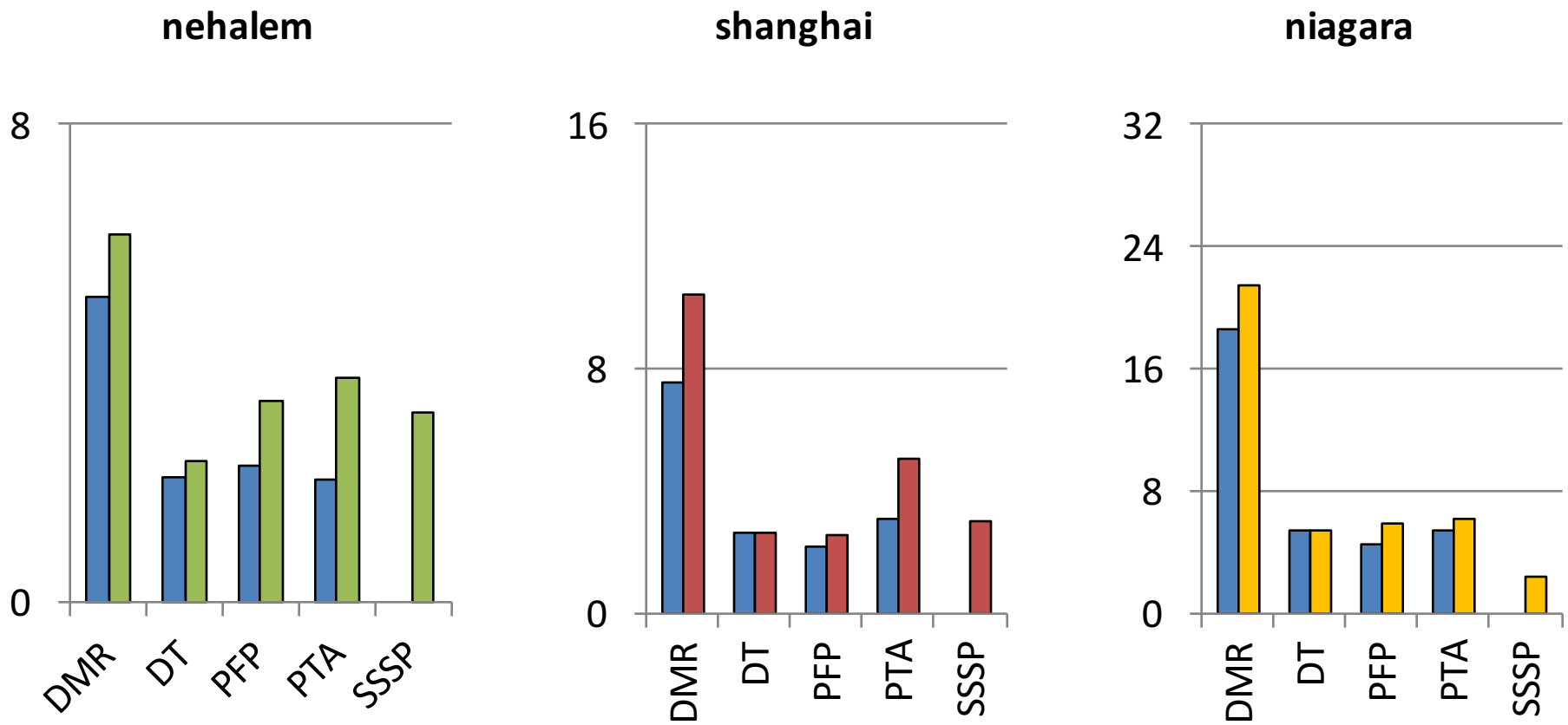
shanghai

Speedup: DMR



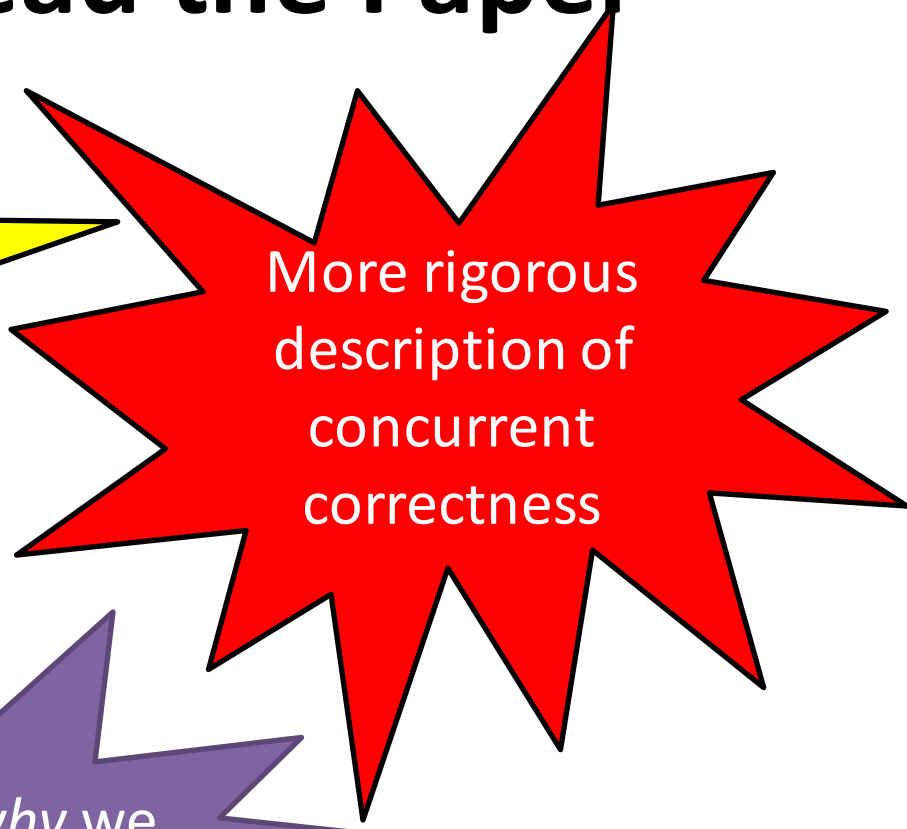
shanghai

Speedup



AS versus WS-F

An Advert to Read the Paper



Conclusion

1. High-level programmer control over scheduling is important for serial ***and*** parallel performance
2. Providing such control is easy and extensible

- Future work
  - Architectural optimizations
  - More algorithms

http://iss.ices.utexas.edu/galois