Parrot + dBug: Fast and Reliable Multithreading

Jiri Simsa

Heming Cui, Yi-Hong Lin, Hao Li, Xinan Xu, Junfeng Yang (Columbia University)
Ben Blum, Garth Gibson, Randy Bryant (Carnegie Mellon University)

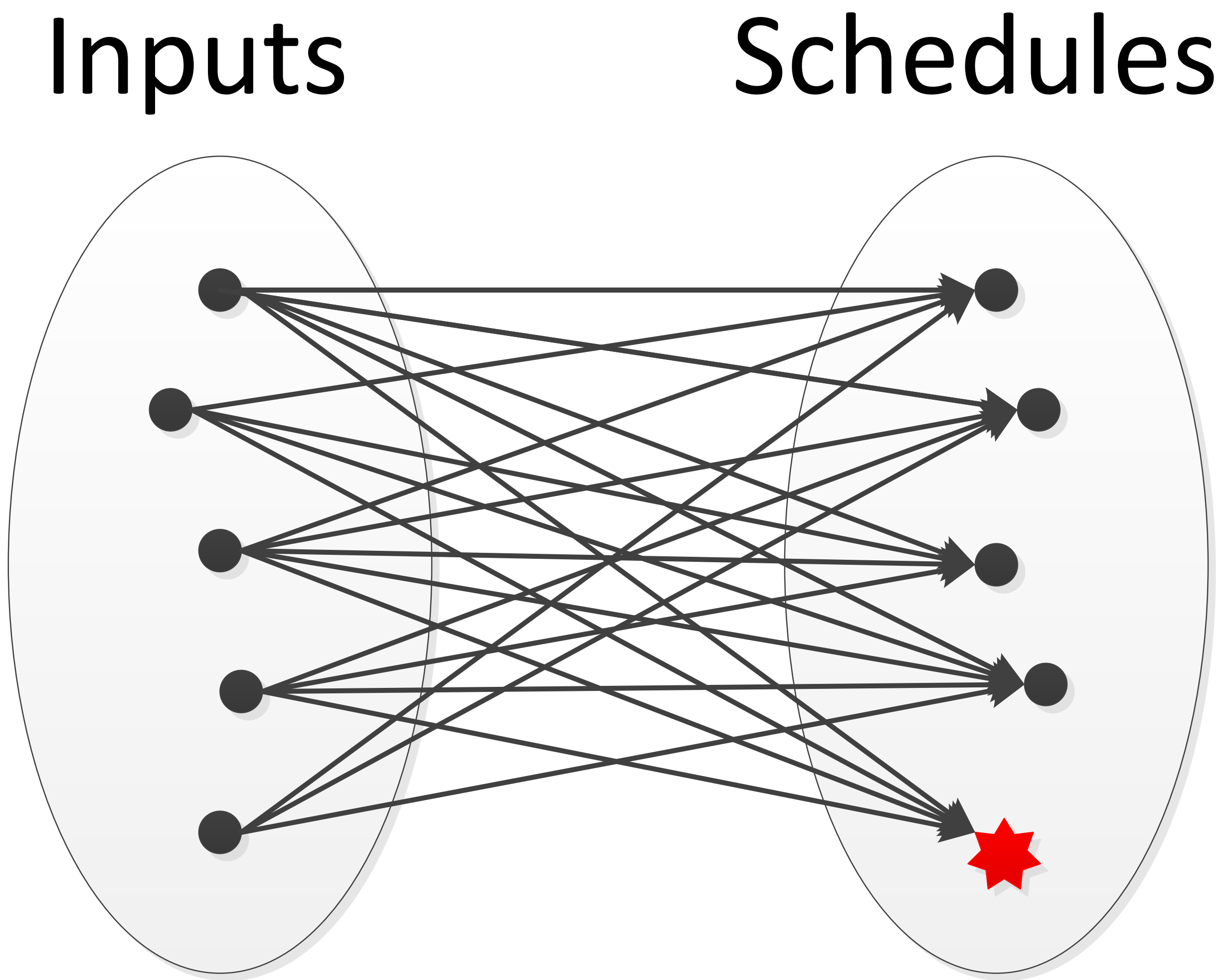
http://www.istc-cc.cmu.edu/

Intel Science & Technology Center for Cloud Computing

Motivation

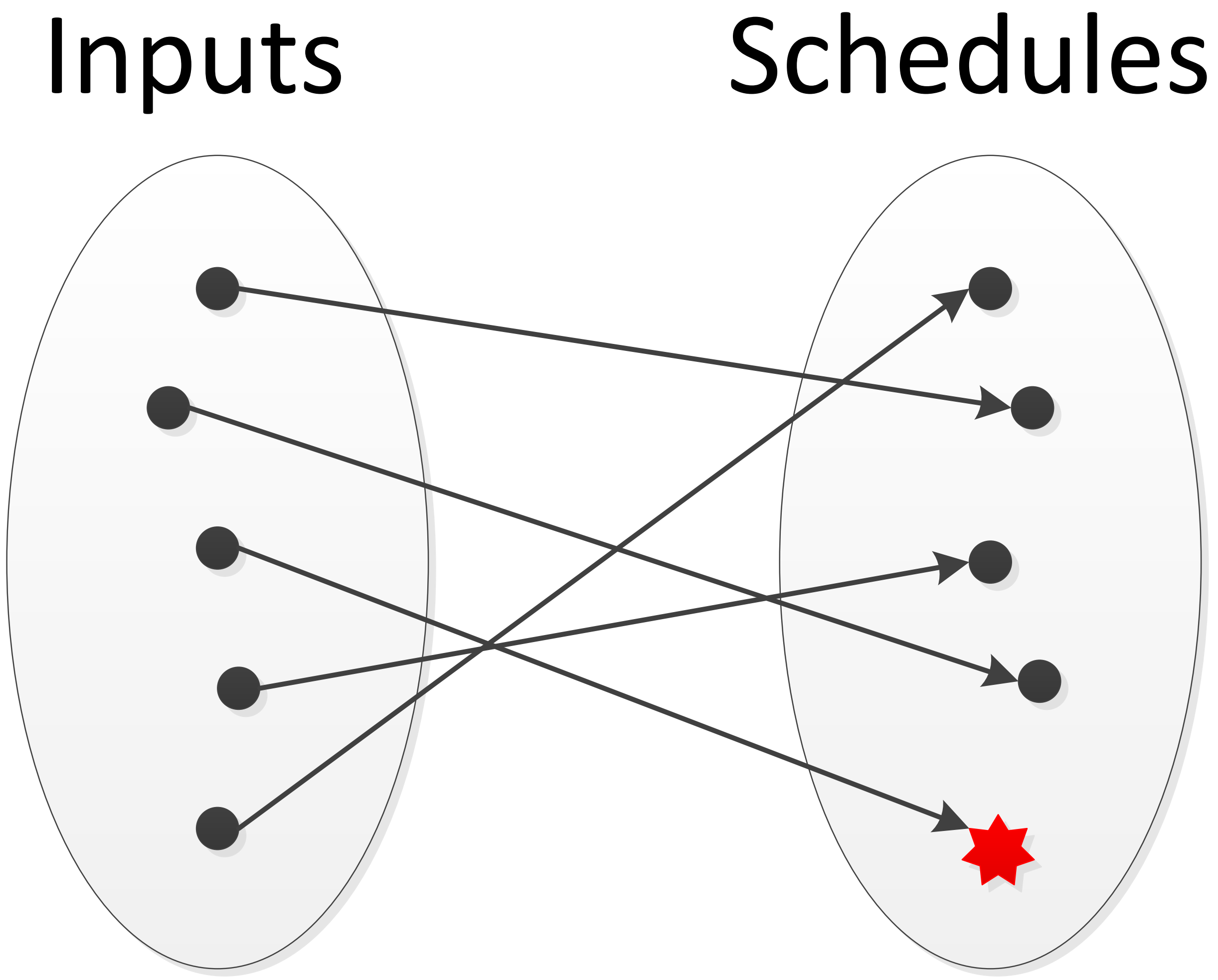
- Clouds of machines are increasingly more concurrent
- Testing and debugging concurrent programs is hard
- Worse still off in the cloud
- Our work:
  - Couples restricted runtime scheduling with systematic testing
  - Improves testability without loss of performance

Runtime Nondeterminism



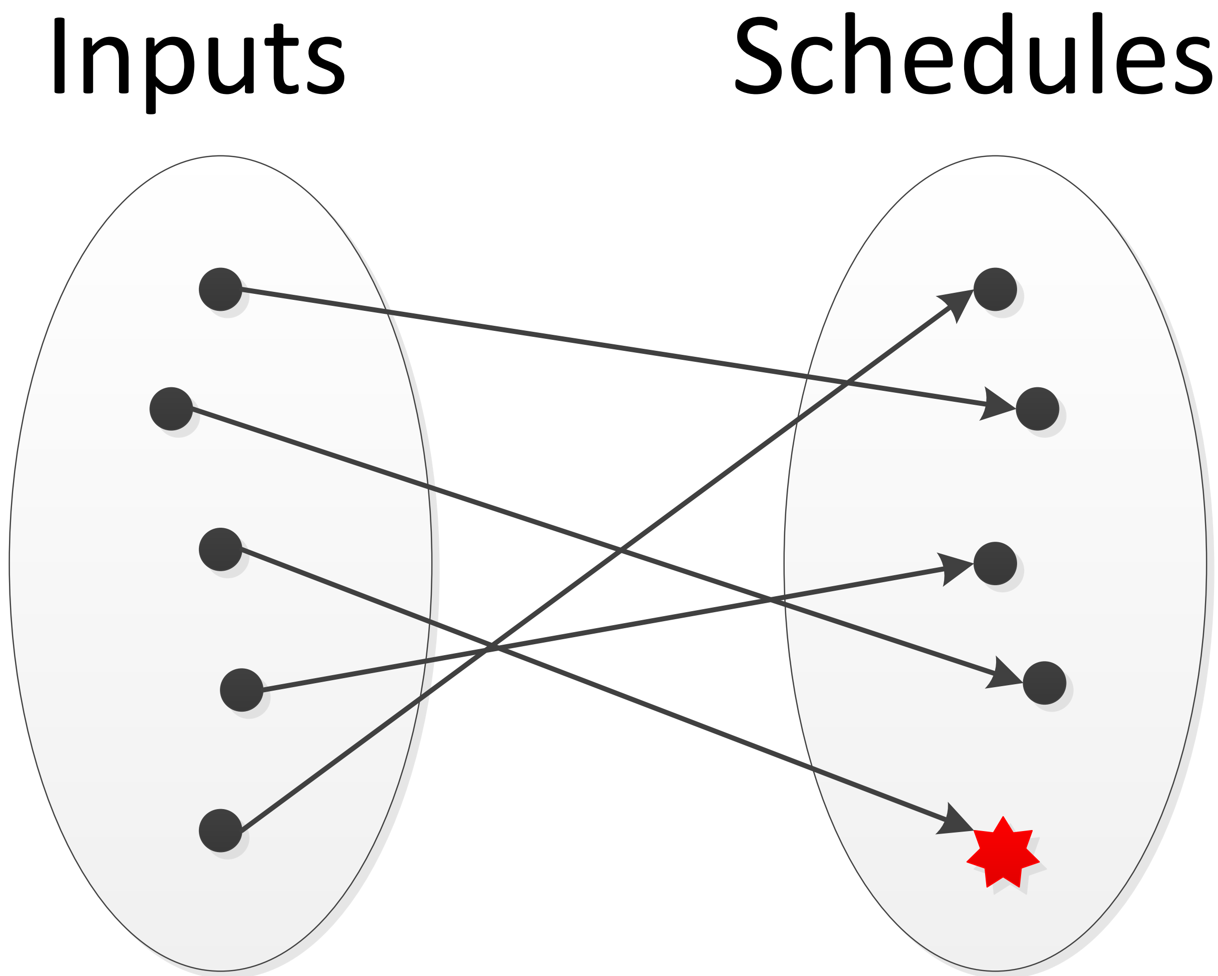
- Combinatorial explosion of number of schedules
- Difficult to reproduce behavior
- Difficult to thoroughly test

Deterministic Multithreading



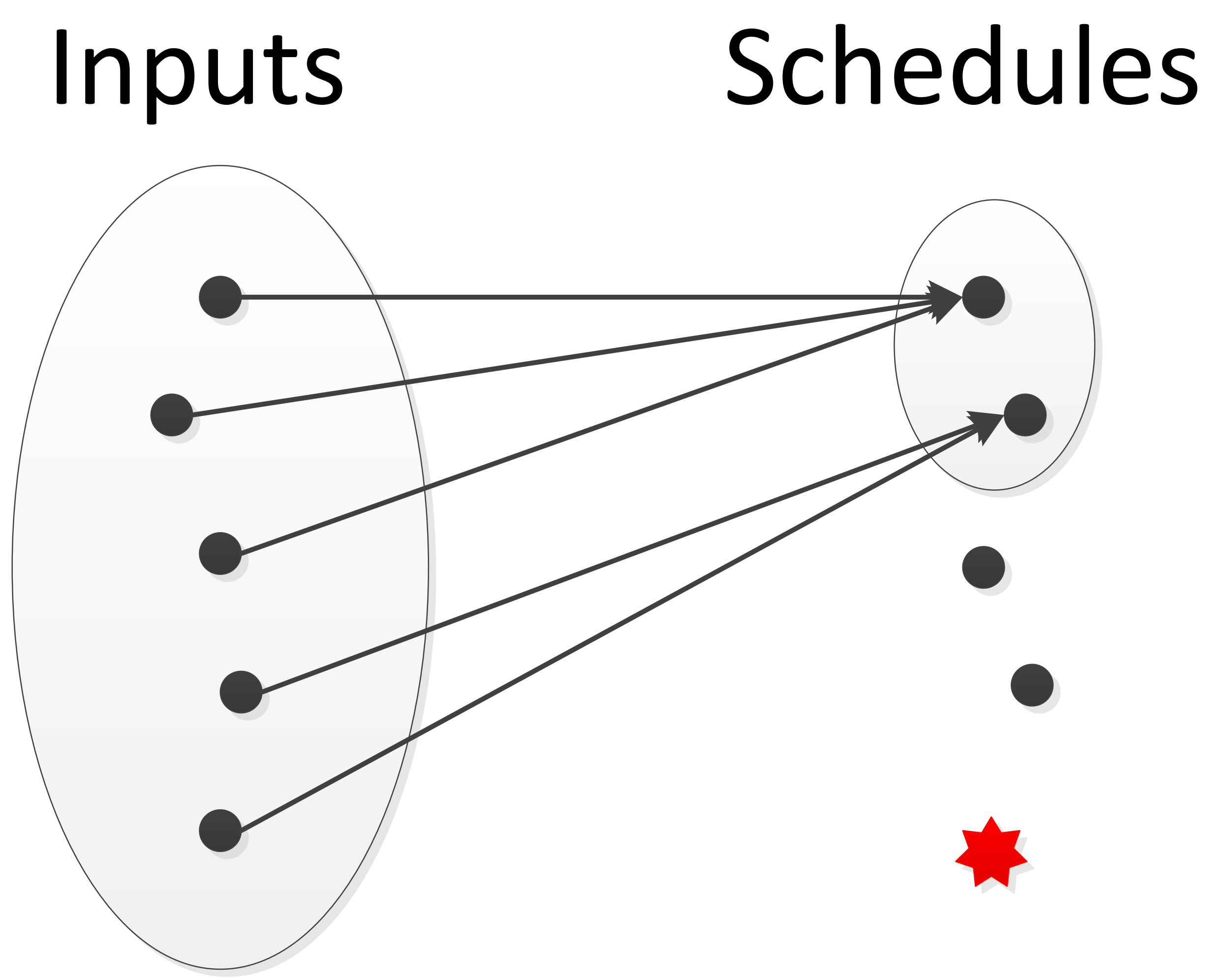
- Deterministically maps inputs to a schedule
- Easy to reproduce behavior
- Difficult to thoroughly test

Deterministic Multithreading



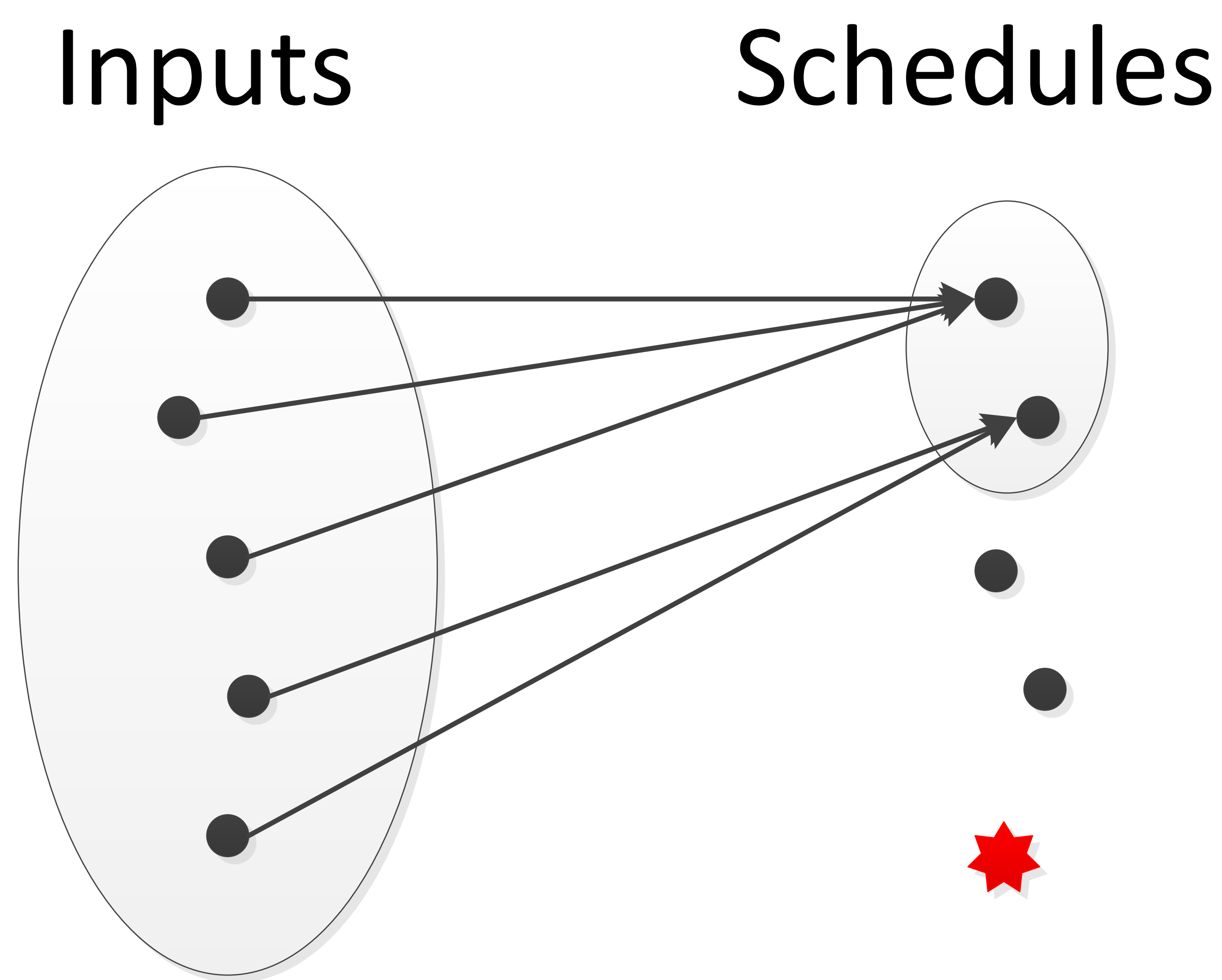
CoreDet[Bergan2010], Determinator[Aviram2010], DMP[Devietti2009], dthreads[Berger2011], Grace[Berger2009], Kendo[Olszewski2009], Peregrine[Cui2011], Tern[Cui2010]

Stable Multithreading



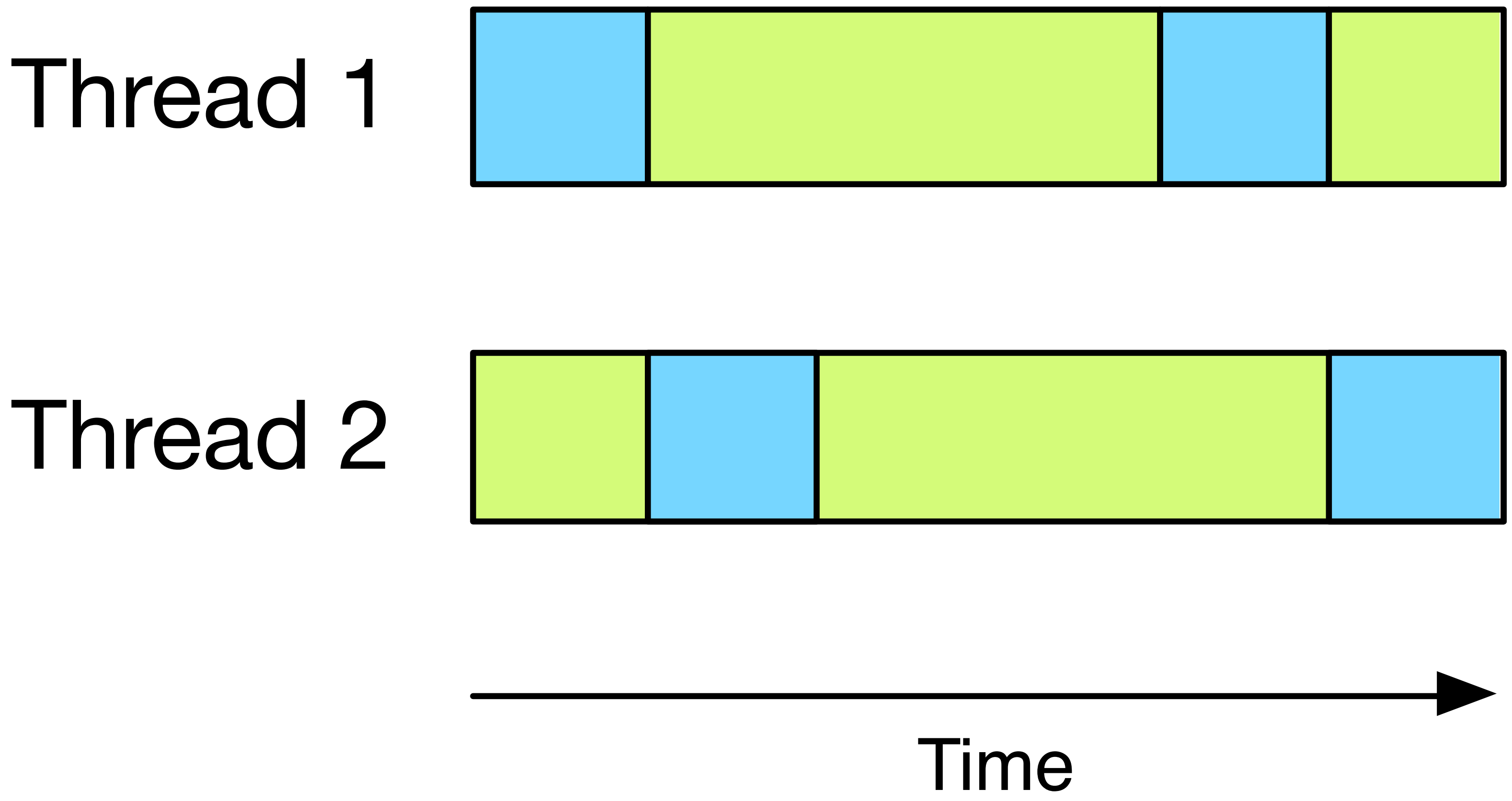
- Maps similar inputs to identical schedules
- Easy to reproduce behavior and thoroughly test
- Slower than non-determinsitic execution

Stable Multithreading

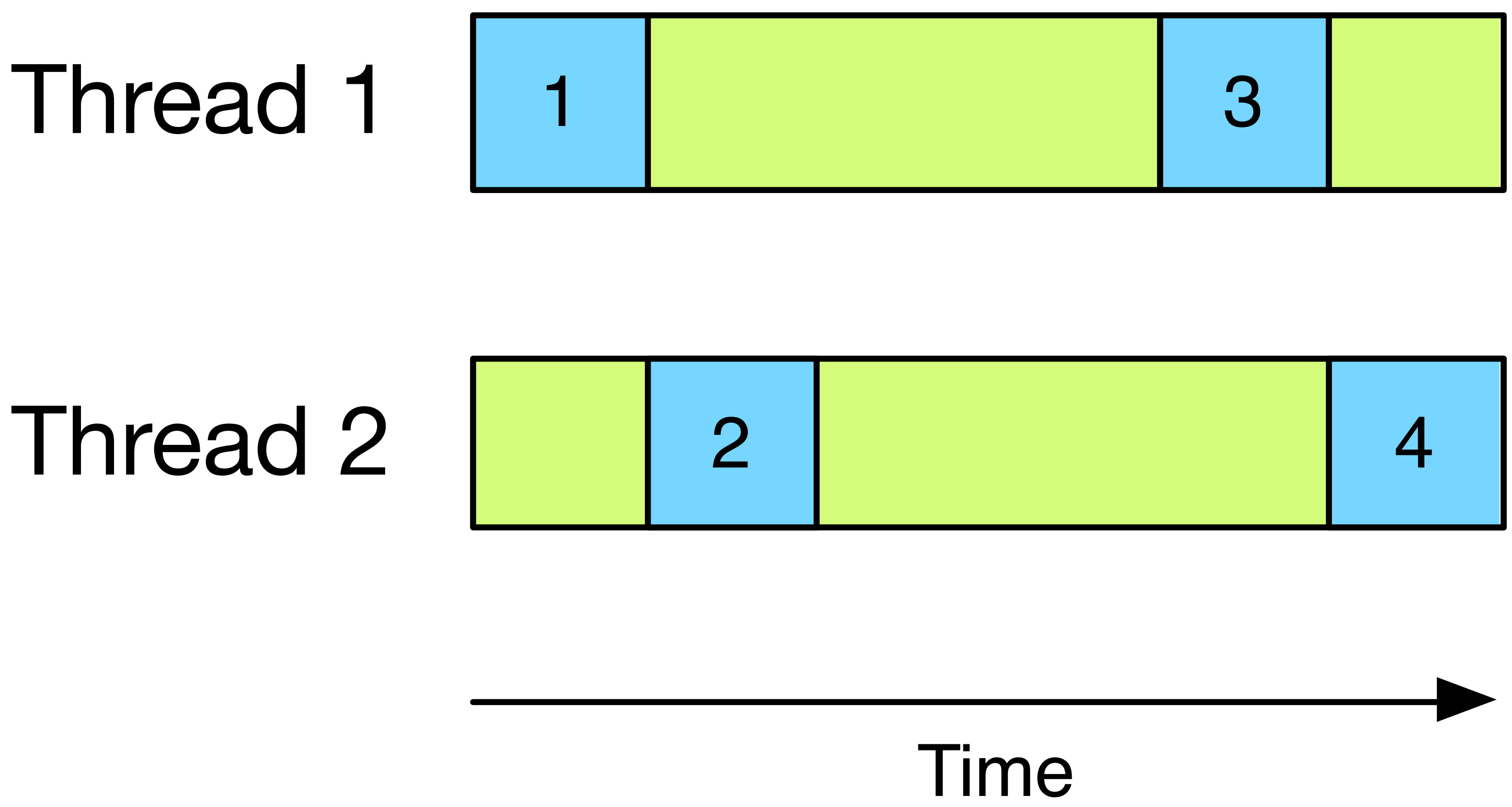


Determinator[Aviram2010], dthreads[Berger2011], Grace[Berger2009], Peregrine[Cui2011], Tern[Cui2010]

What Is a Schedule?

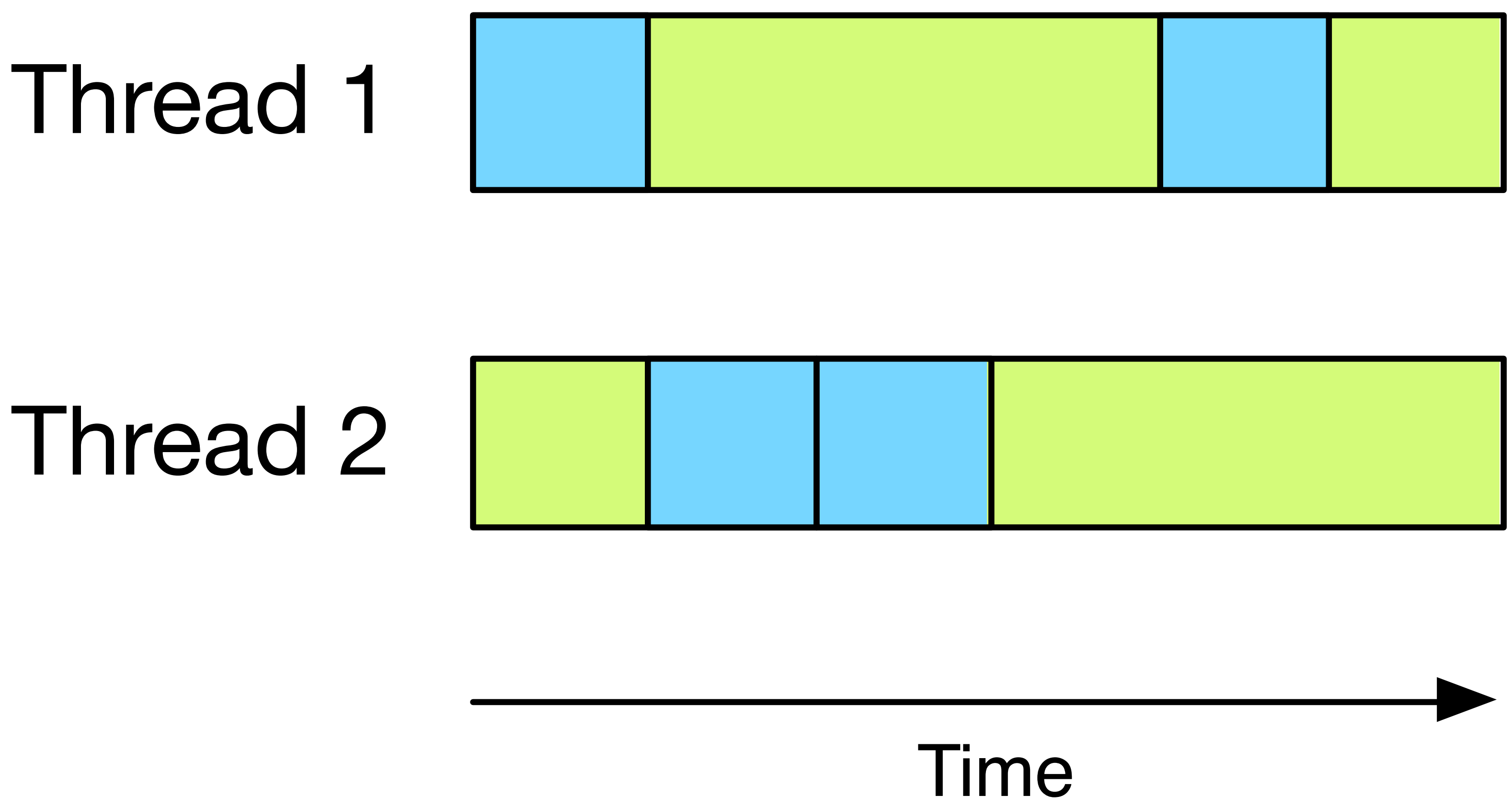


What Is a Schedule?

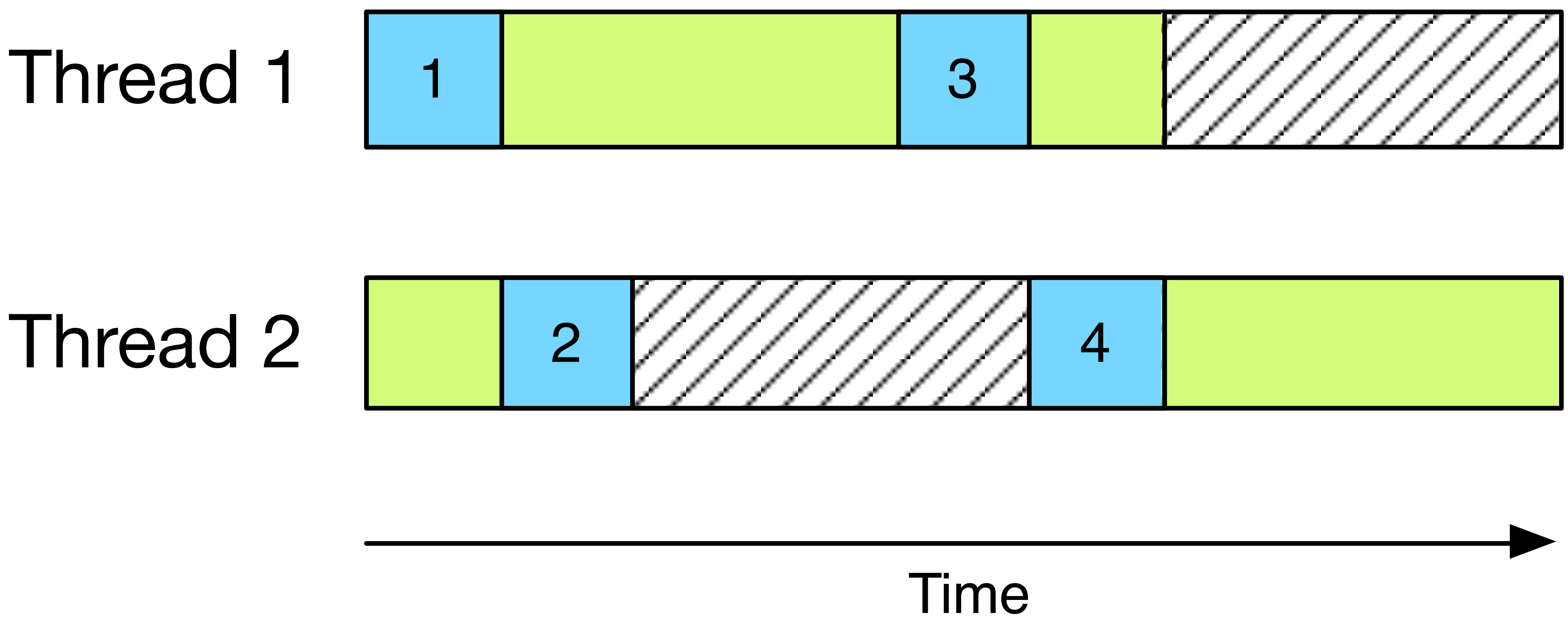


- Ordering of concurrent events
- Shared memory accesses ⇨ strong determinism
- Synchronization events ⇨ weak determinism

Serialization Problem



Serialization Problem



- Timing of concurrent events depends on input
- Stable multithreading reuses schedules
- Artificial serialization of computation

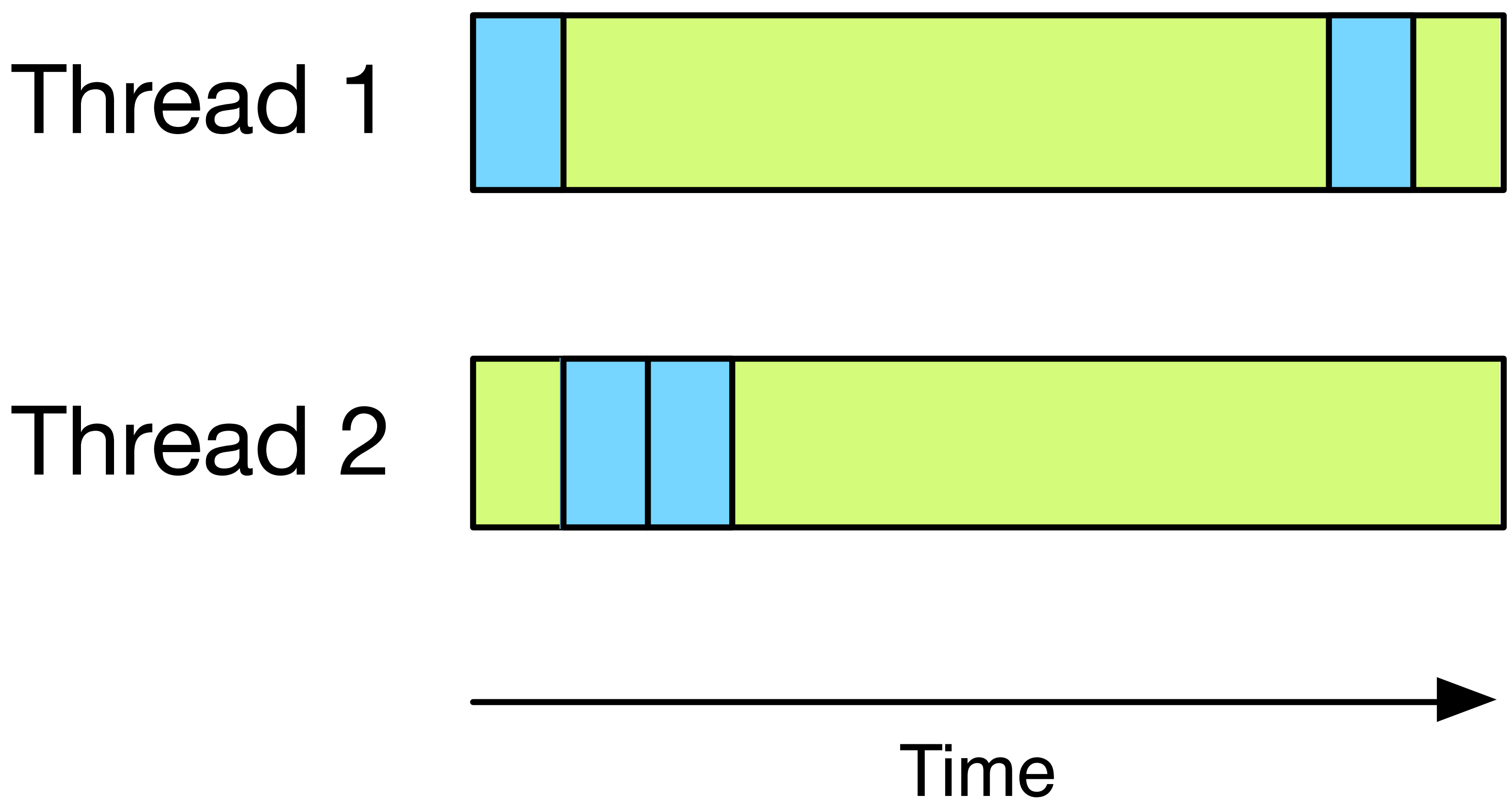
Outline

- Motivation
- Performance Hints
  - Soft Barrier
  - Performance-Critical Section
- Parrot Runtime Environment
- dBug Testing Environment
- Evaluation

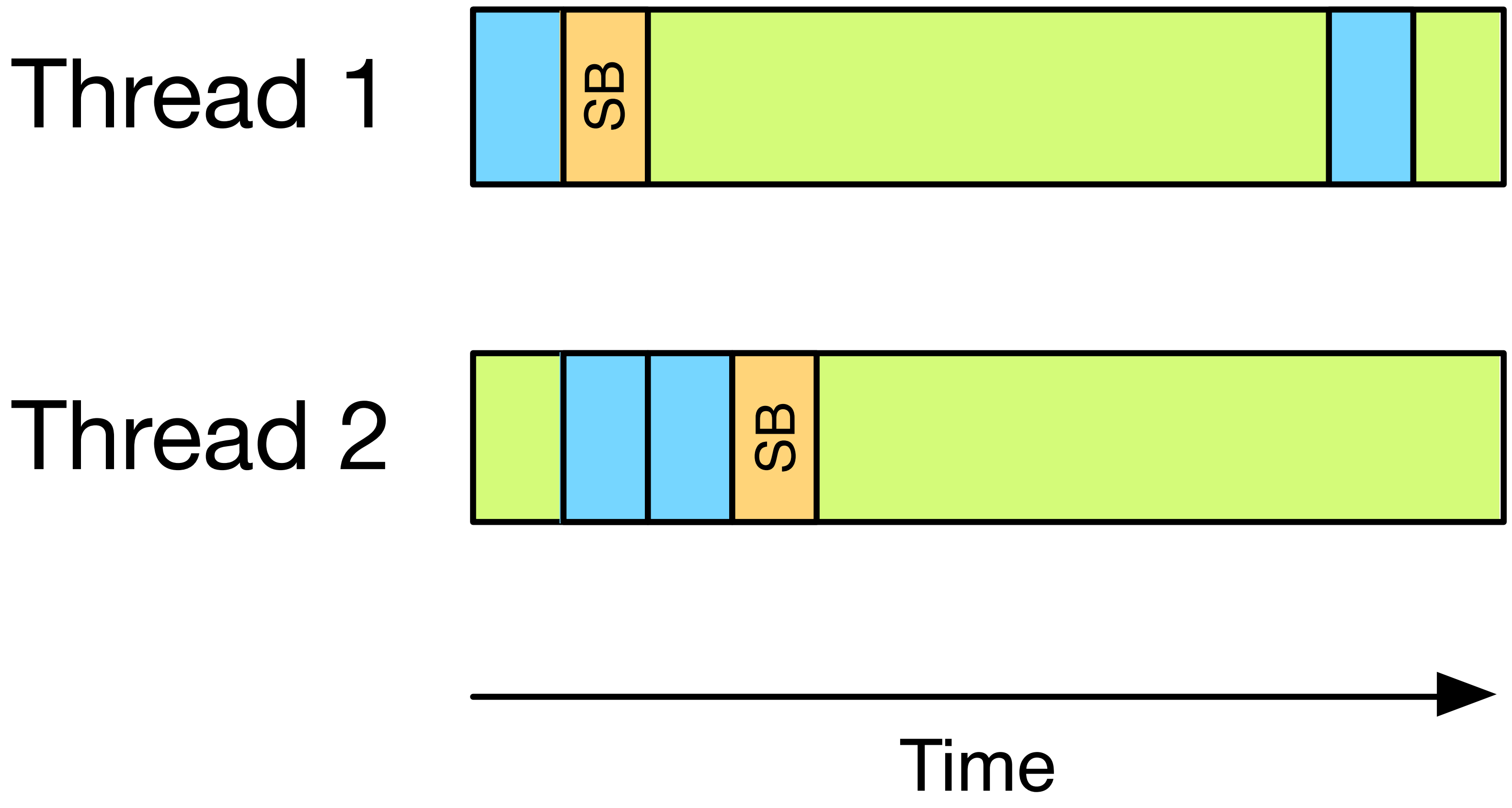
Performance Hints

- Nondeterministic Runtime ⇨ too many schedules
- Stable Multithreading ⇨ too few schedules
- Neither of them offers scheduling interface
- Our solution ⇨ performance hints
- Simple API to specify efficient schedules

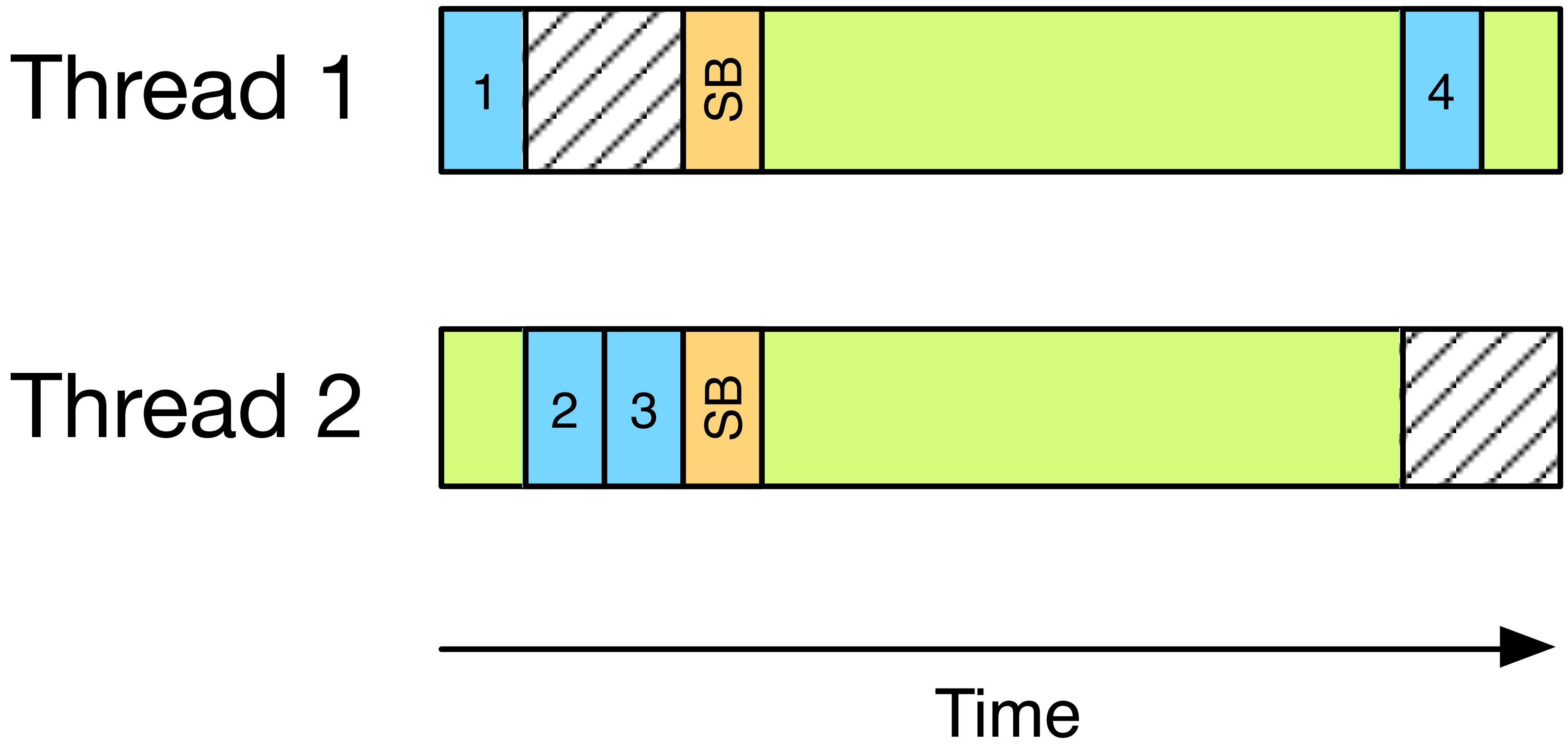
Soft Barrier



Soft Barrier

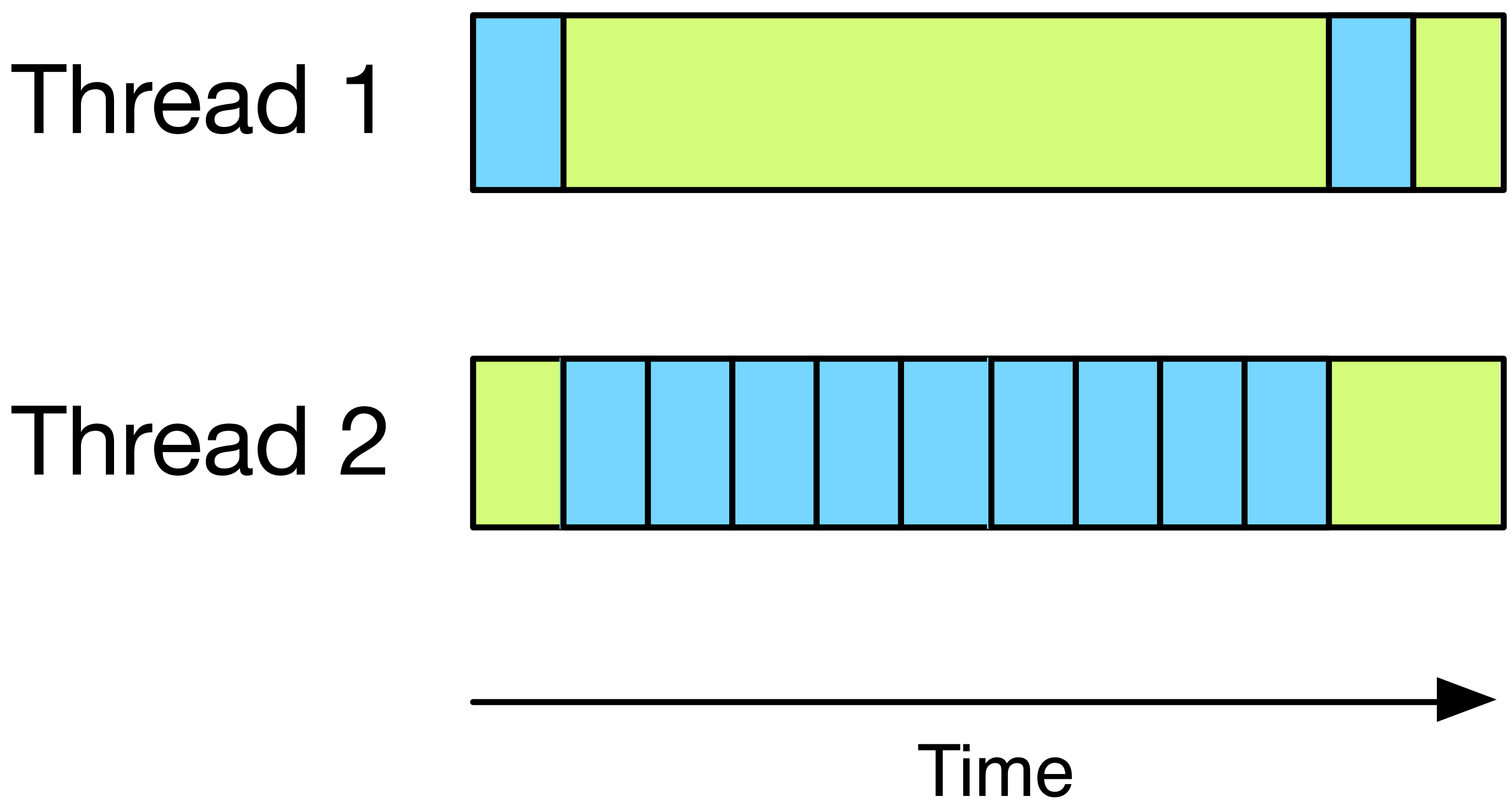


Soft Barrier

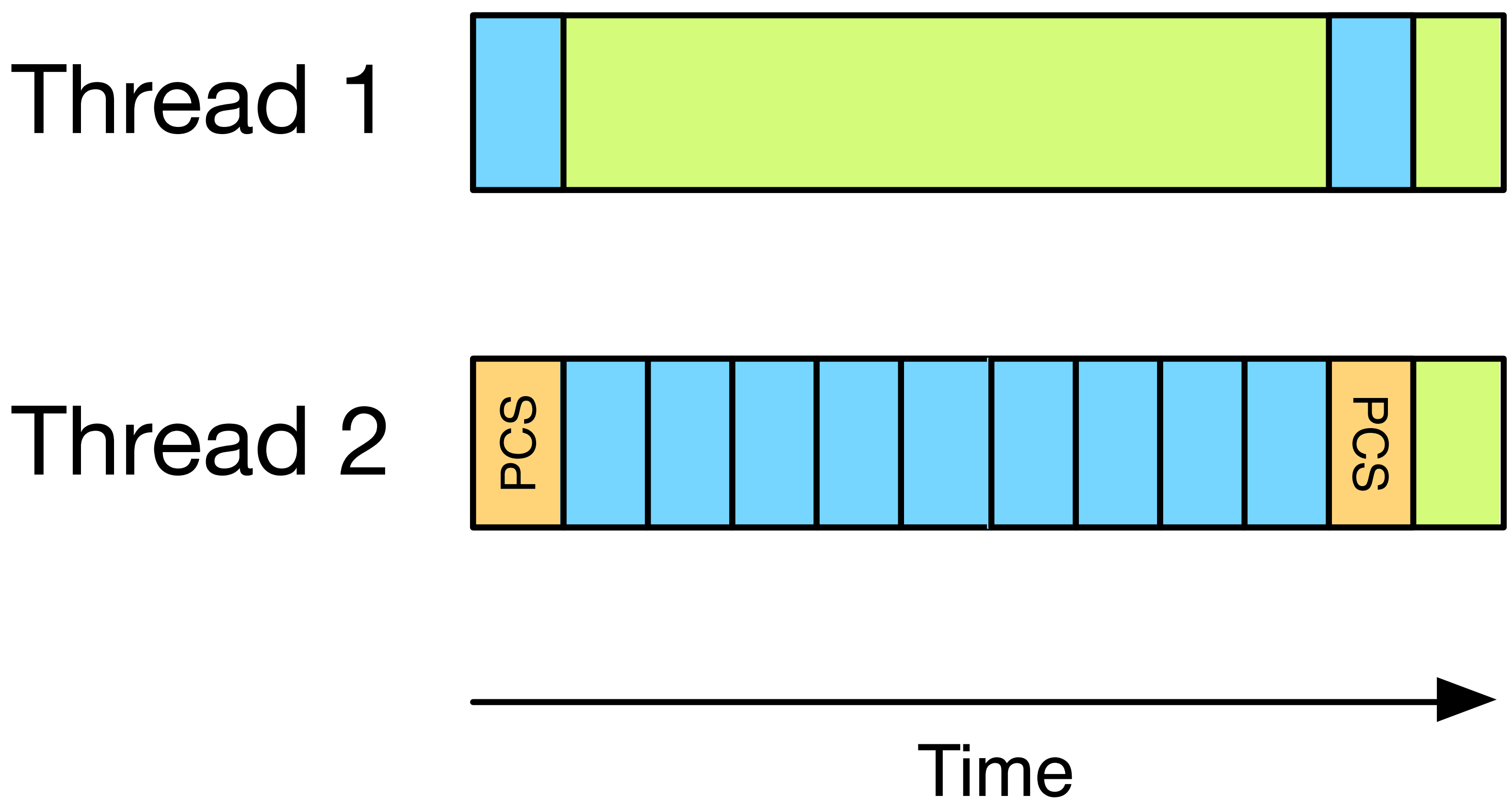


- Expresses co-scheduling intent
- Unlike traditional barrier, waiting can time out
- Does not introduce non-determinism

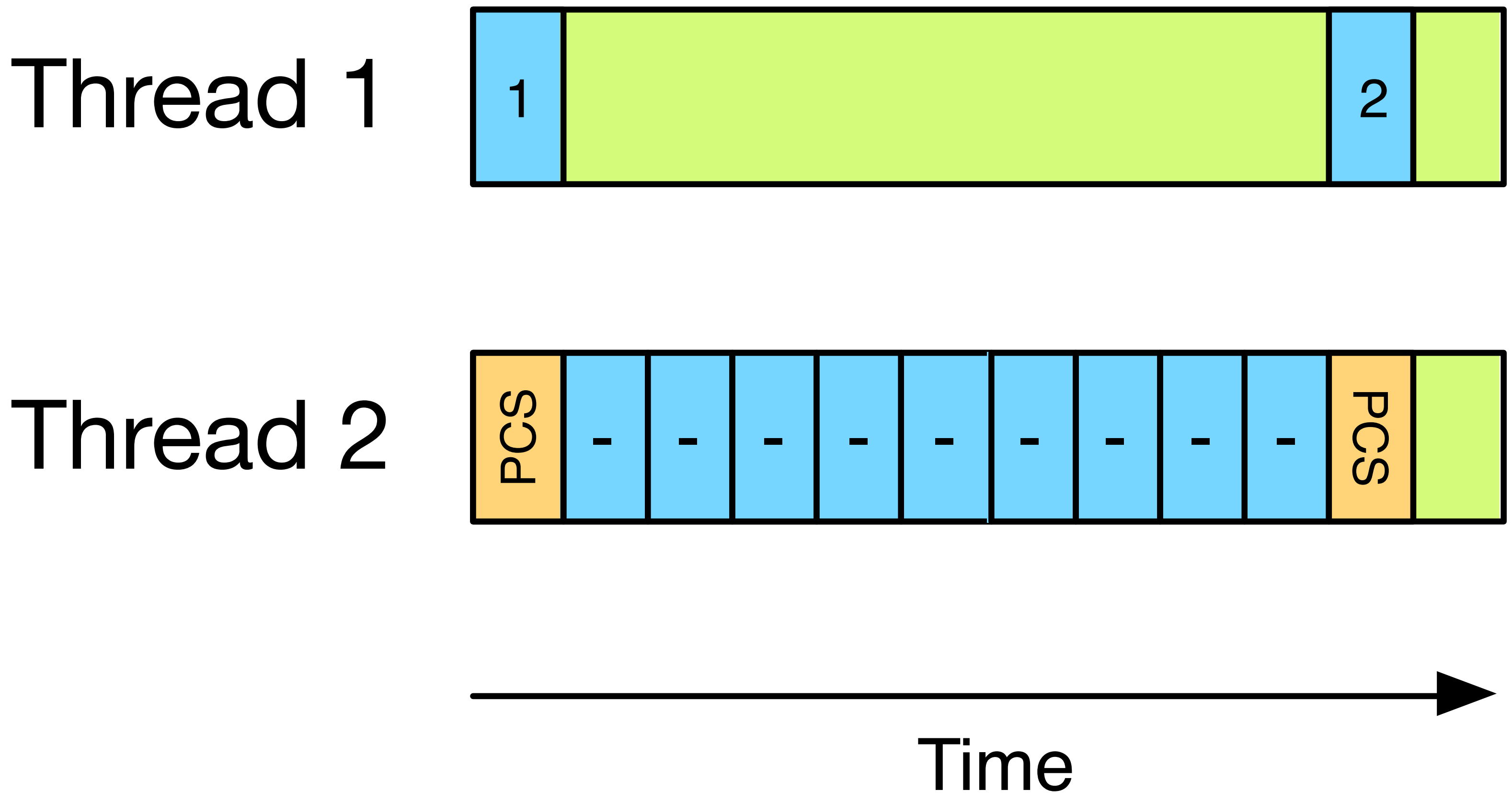
Performance-Critical Section



Performance-Critical Section



Performance-Critical Section

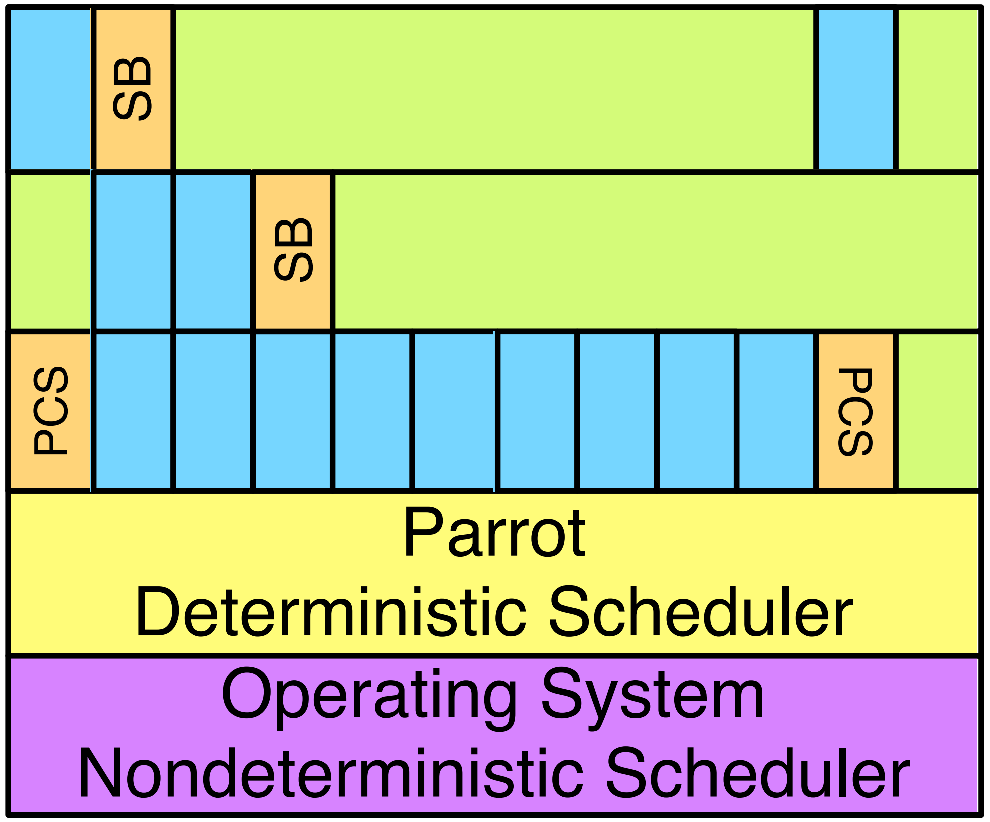


- Identifies a potential performance bottleneck
- Disables ordering of concurrent events
- Does introduce non-determinism

Outline

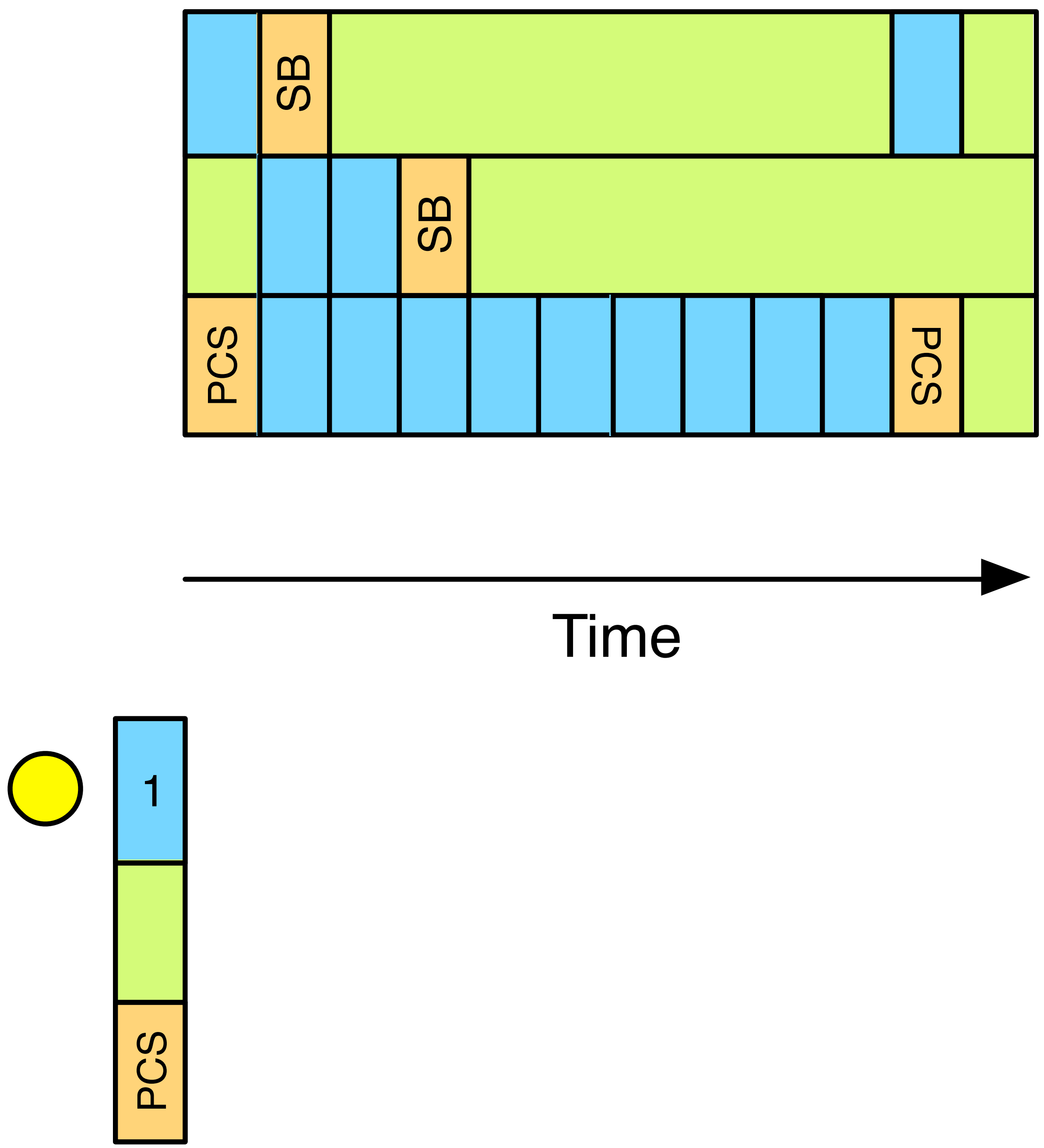
- Motivation
- Performance Hints
- Parrot Runtime Environment
  - Architecture
  - Execution Example
- dBug Testing Environment
- Evaluation

Parrot Architecture

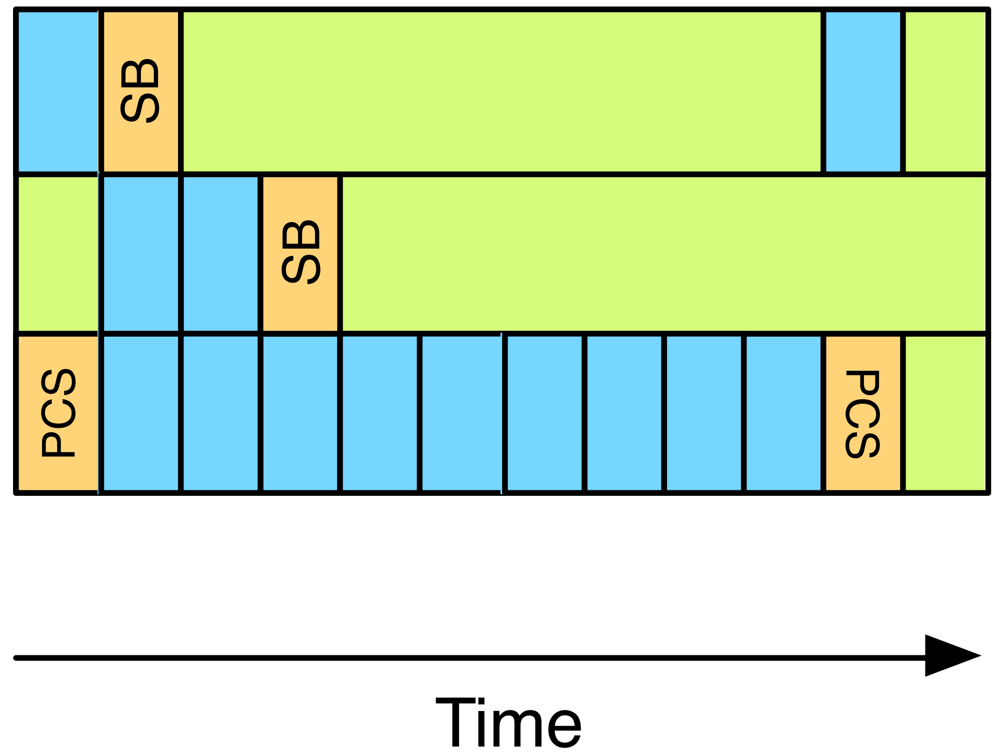


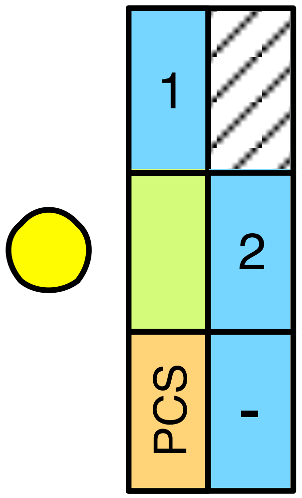
- Parrot interposes on POSIX interface
- Default round-robin ordering of synchronizations
- Hints can be used to tune performance

Parrot Execution Example

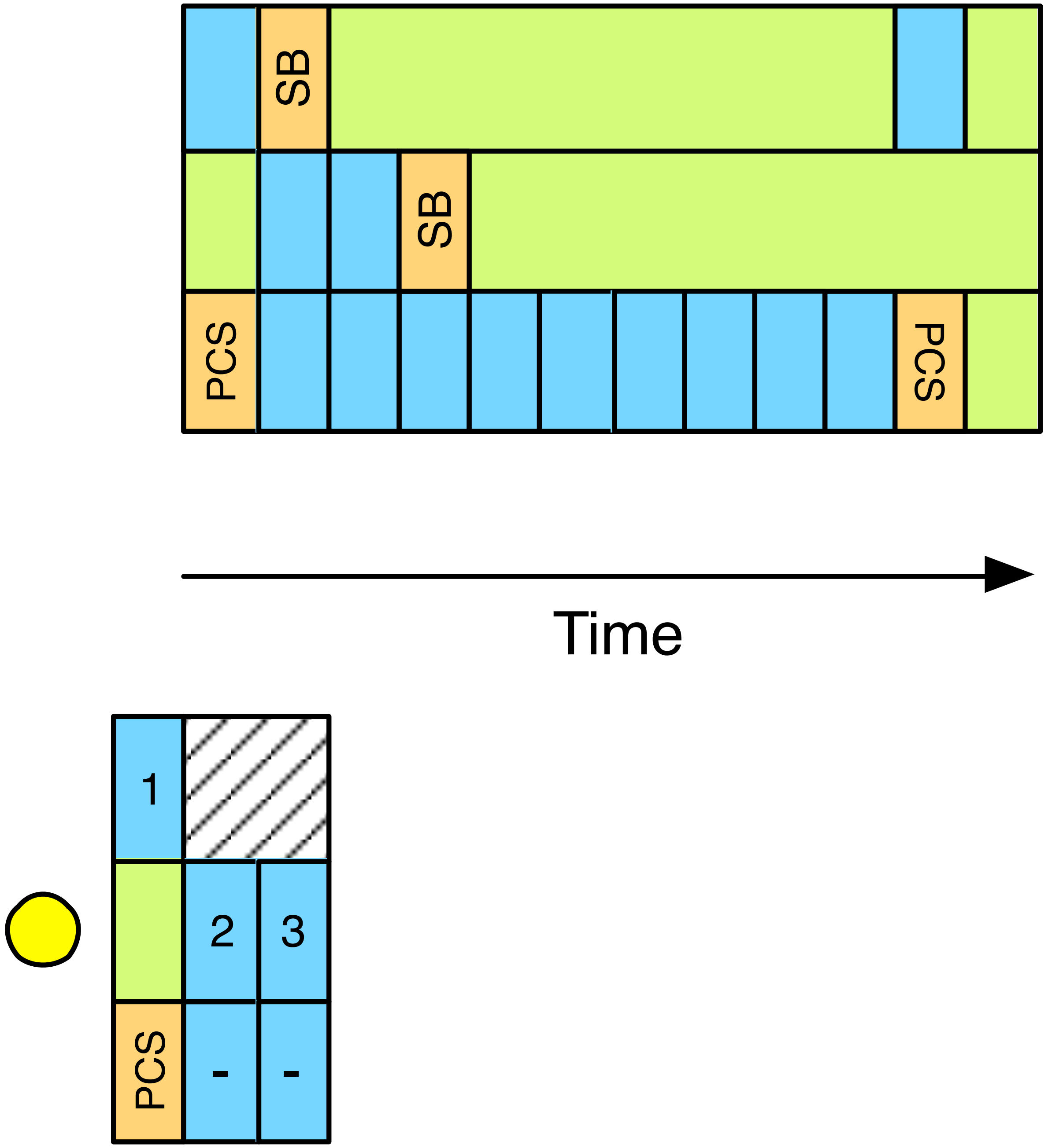


Parrot Execution Example

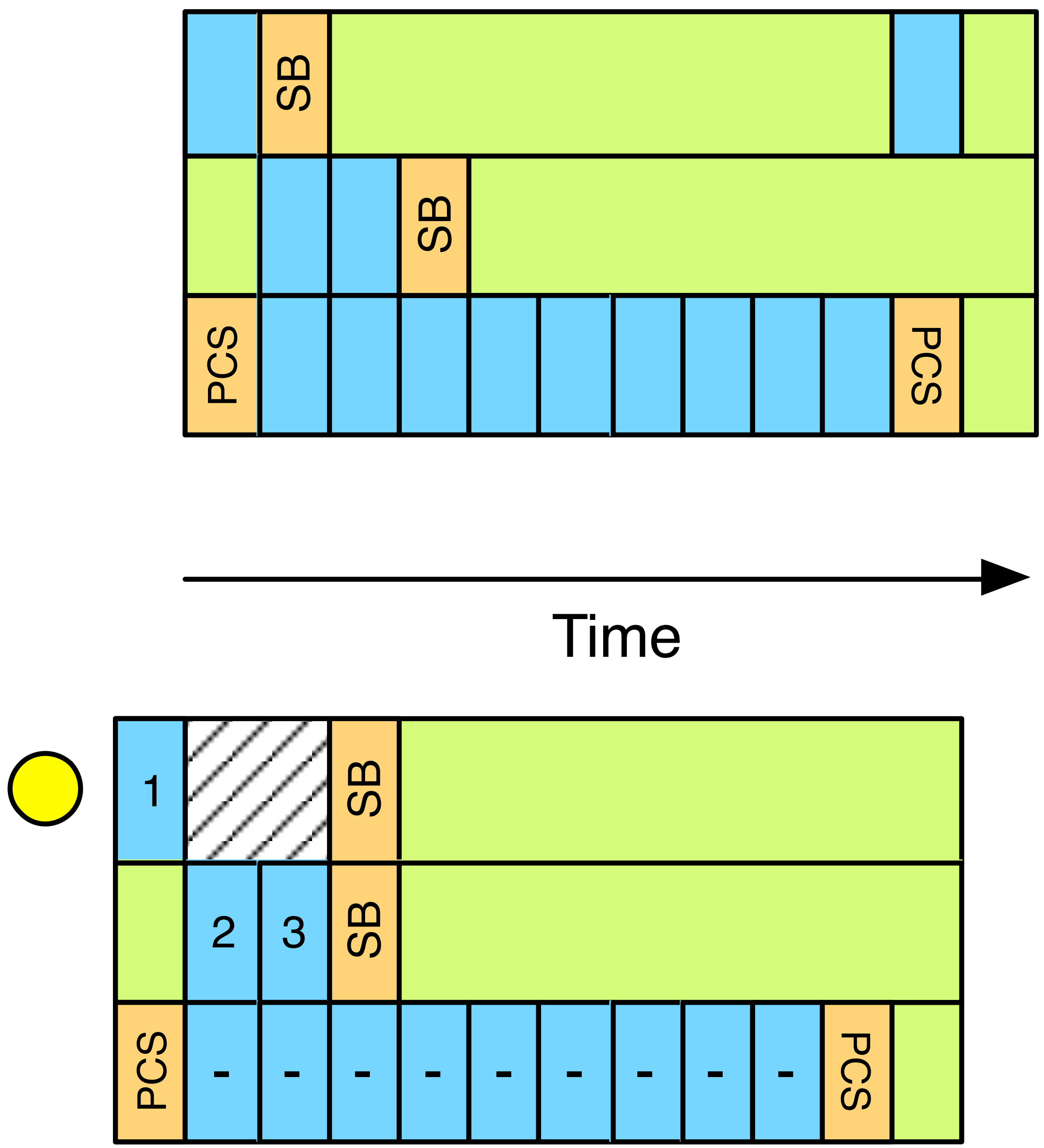




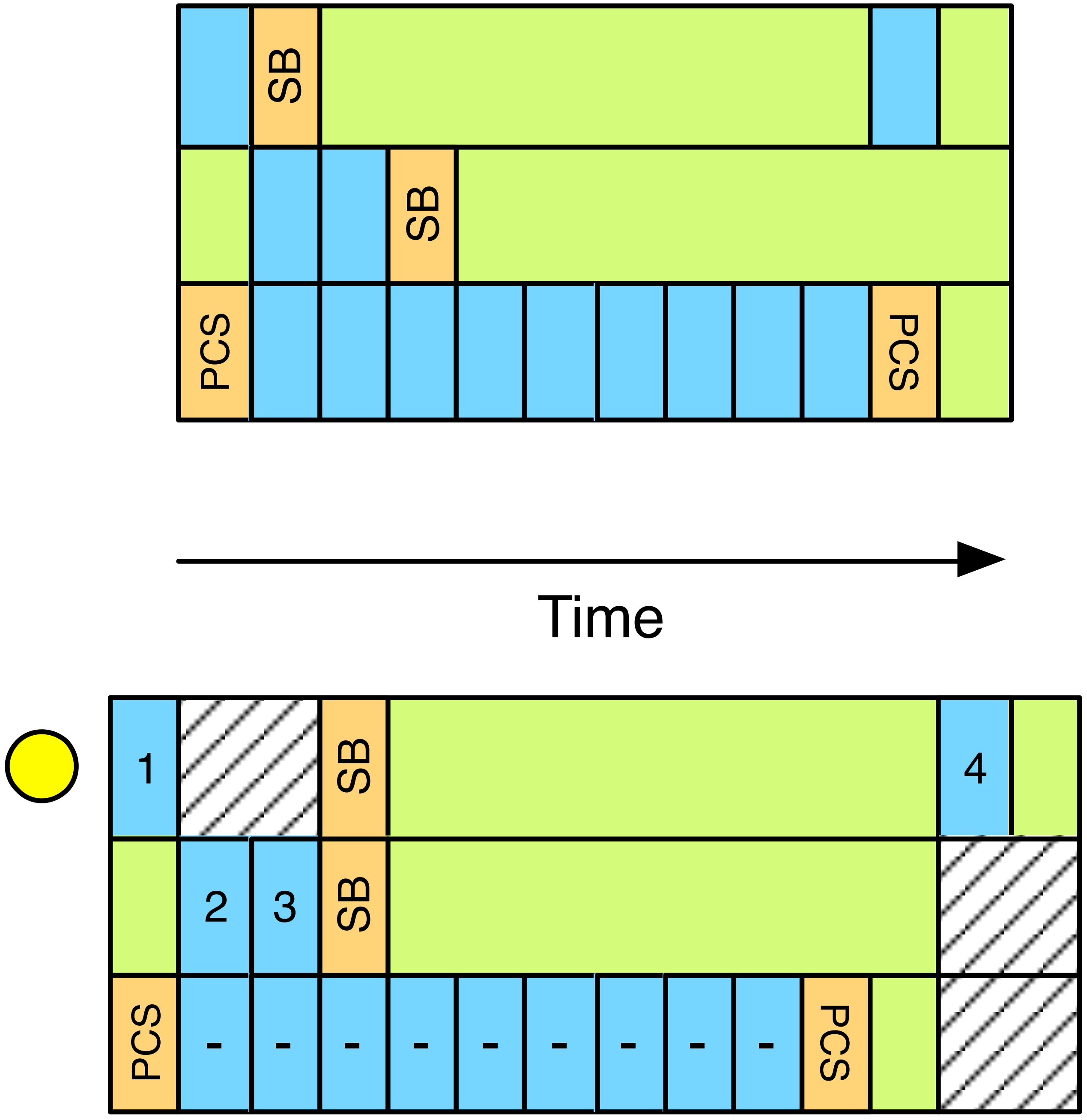
Parrot Execution Example



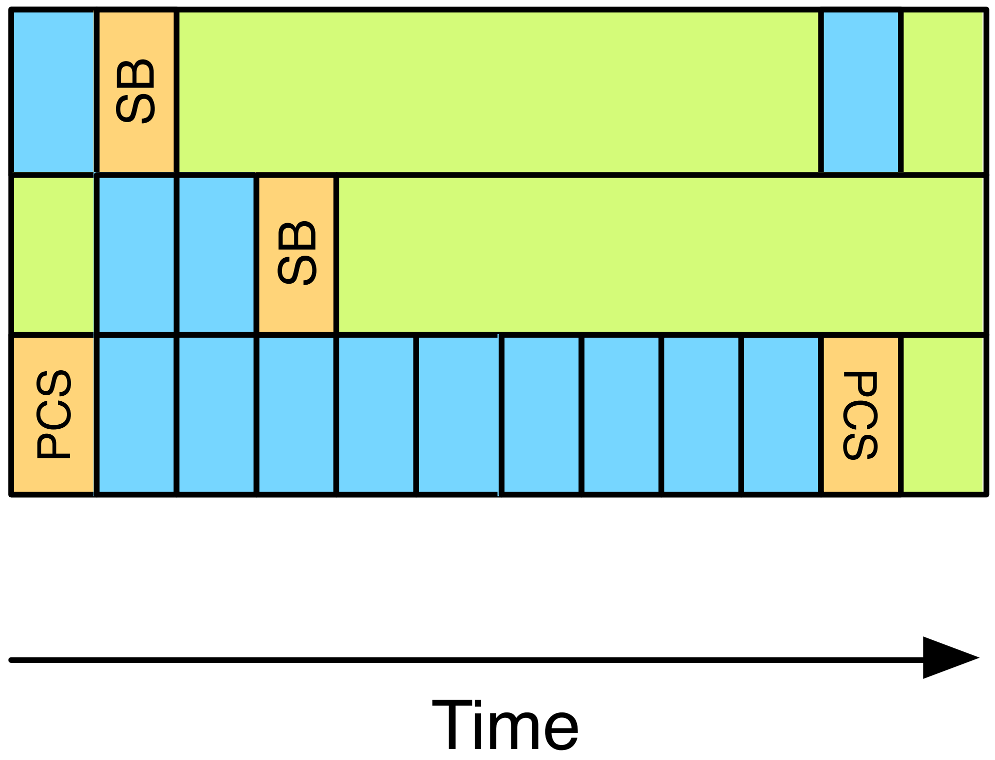
Parrot Execution Example



Parrot Execution Example



Performance vs. Testability

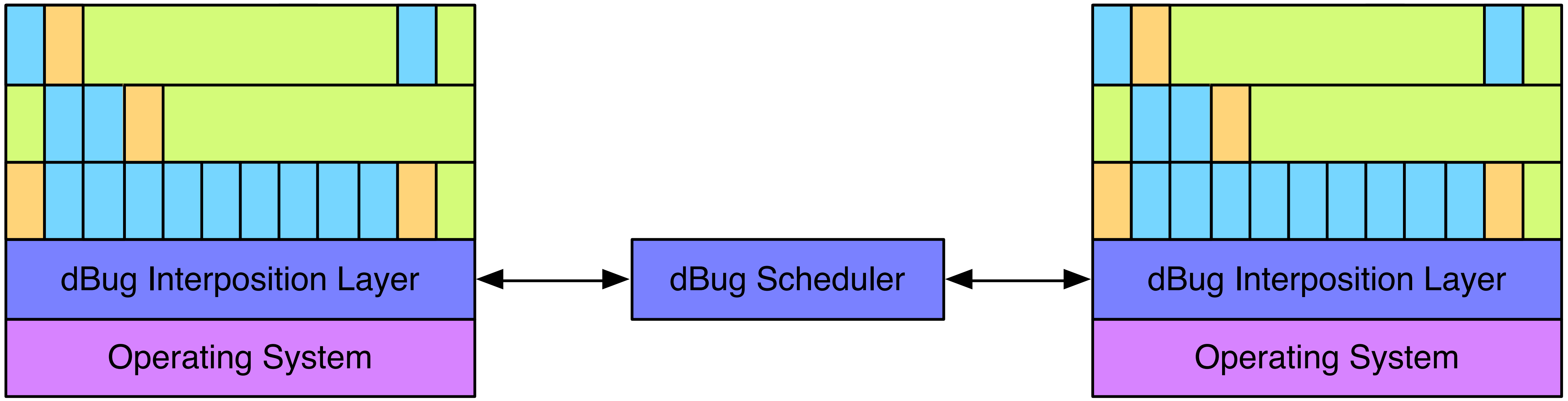


- Without Parrot ⇨ many schedules and fast
- With Parrot and no hints ⇨ one schedule but slow
- With Parrot and hints ⇨ few schedules and fast

Outline

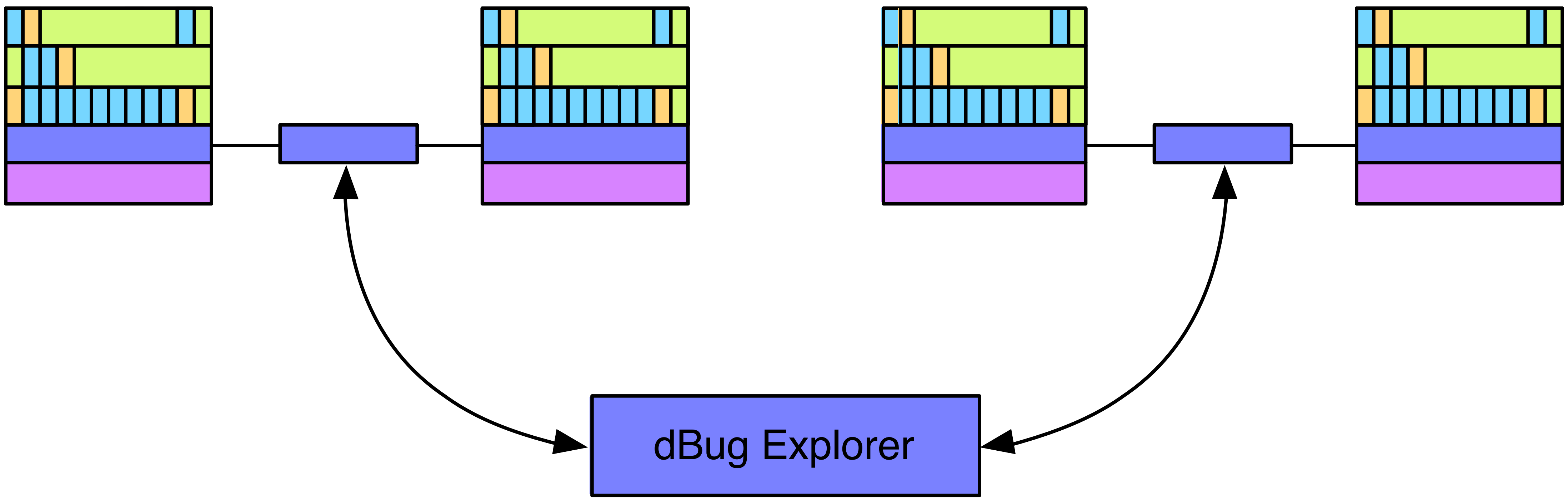
- Motivation
- Performance Hints
- Parrot Runtime Environment
- dBug Testing Environment
  - Architecture
  - Integration with Parrot
- Evaluation

dBug Testing Environment



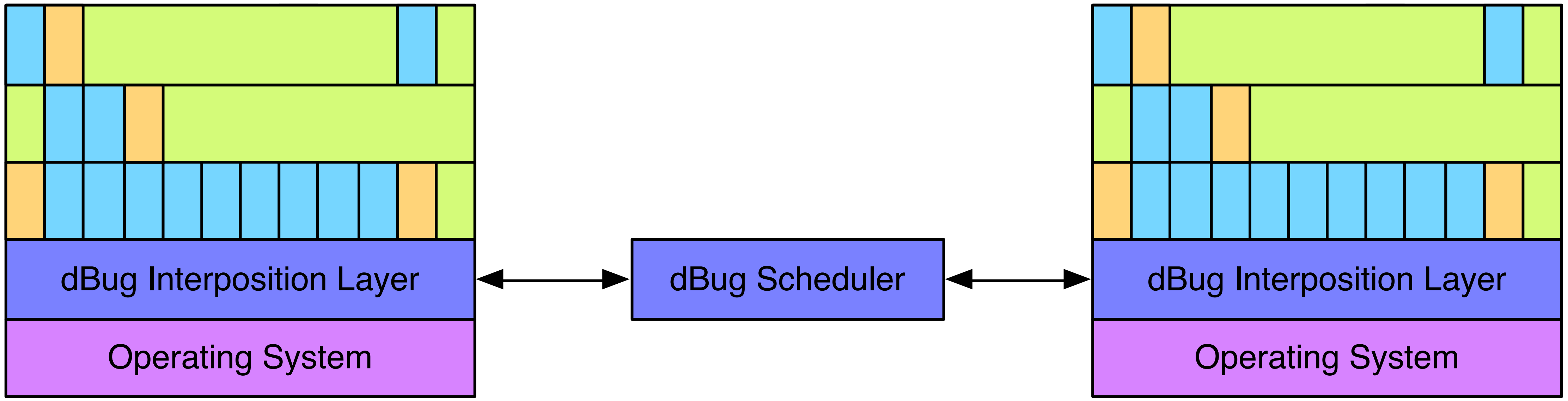
- dBug[Simsa2011] interposes on POSIX interface
- Serializes concurrent program transitions
- Program transitions delimited by synchronizations

dBug Testing Environment

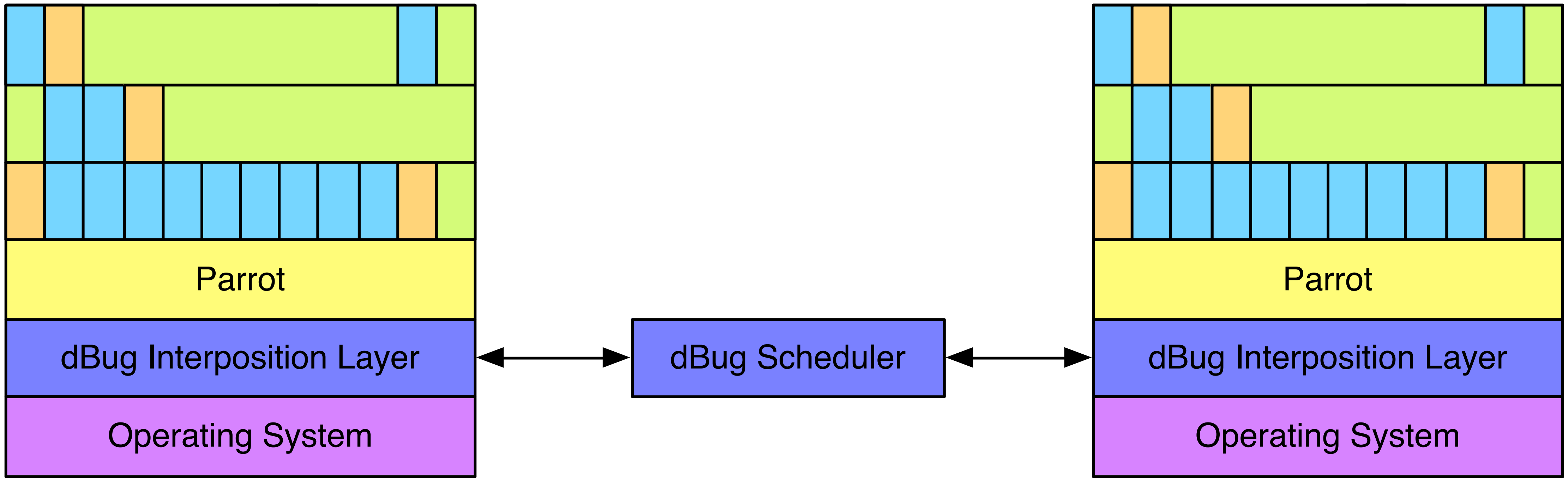


- Explorer repeatedly starts an execution, exploring different schedules of concurrent program transitions
- Uses state space reduction and state space estimation

Parrot and dBug Integration



Parrot and dBug Integration



- Parrot limits nondeterminism exposed to dBug
- dBug only explores schedules allowed by Parrot
- 350 lines of code (250 in Parrot, 100 in dBug)

Outline

- Motivation
- Performance Hints
- Parrot Runtime Environment
- dBug Testing Environment
- Evaluation
  - Performance of Parrot
  - Testing Coverage of dBug

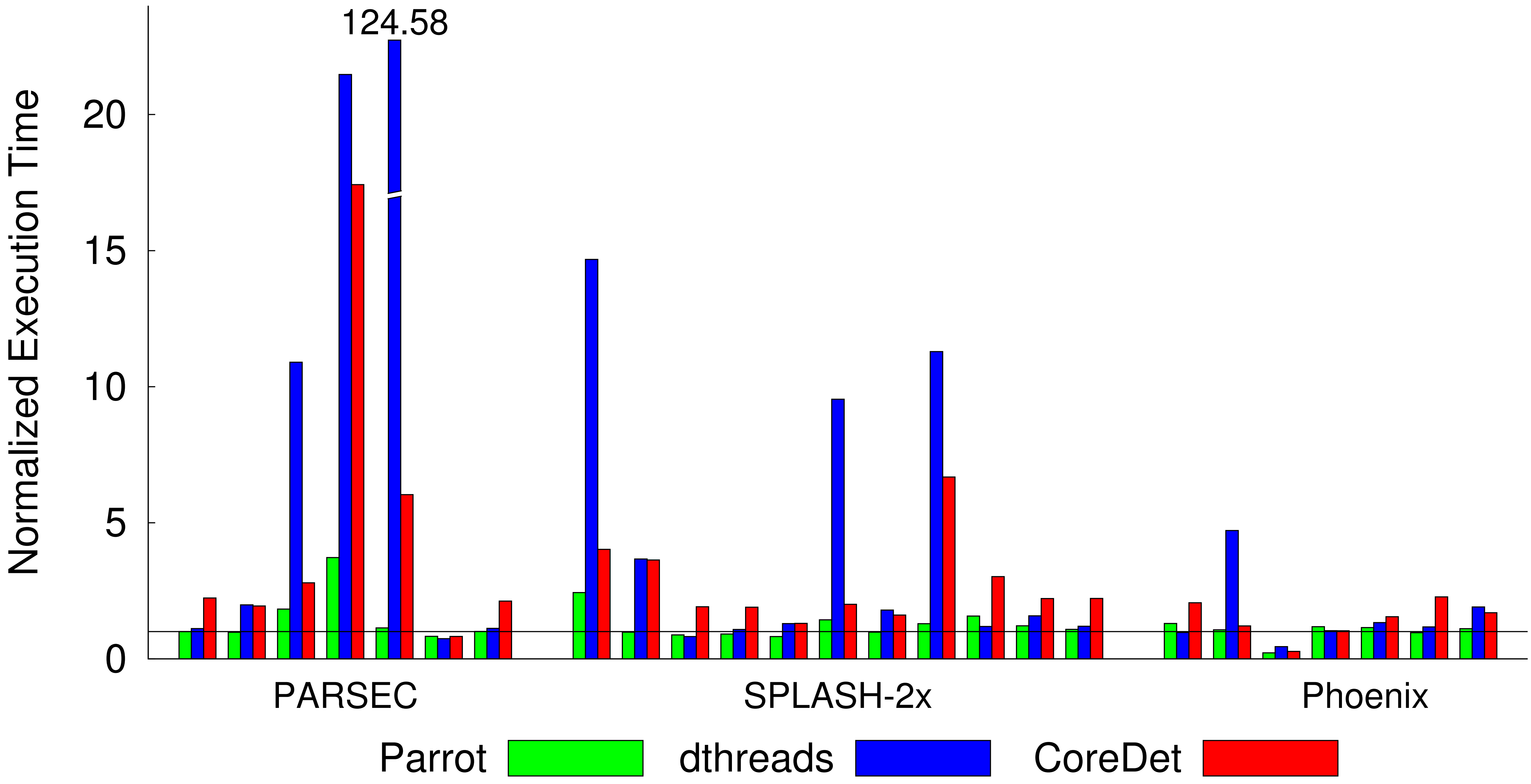
Evaluation Suite

- Real-world workloads:
  - Multiprocess: OpenLDAP, Redis, aget + mongoose
  - Multithreaded: MPlayer, PBZip2, pfscan, BerkeleyDB
- Parsec benchmark (15 workloads)
- Phoenix benchmark (15 workloads)
- Splash benchmark (14 workloads)
- NAS Parallel benchmark (10 workloads)
- ImageMagick image processing utilities (14 workloads)
- Parallel STL algorithm implementations (33 workloads)

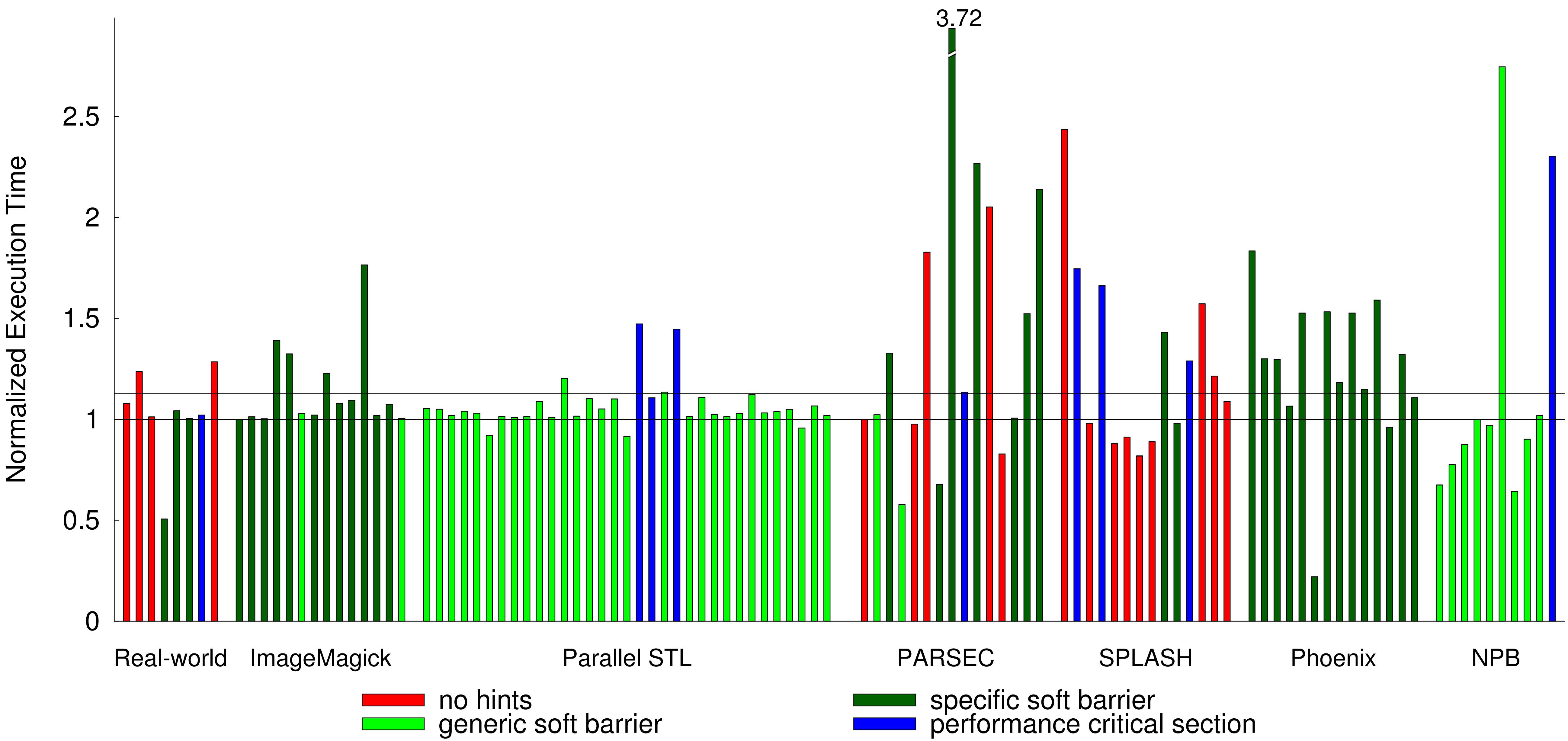
Experimental Setup

- 2.80 GHz Intel Xeon with 24 cores, 64 GB memory
- Linux 3.2.14
- Performance measurements (Parrot):
  - Use between 8 and 24 threads and large inputs
  - Repeated 10-100x to bring standard deviation below 1%
- Testing measurements (Parrot + dBug):
  - Use 2 threads and small inputs
  - State space estimates based on 24 hour runs

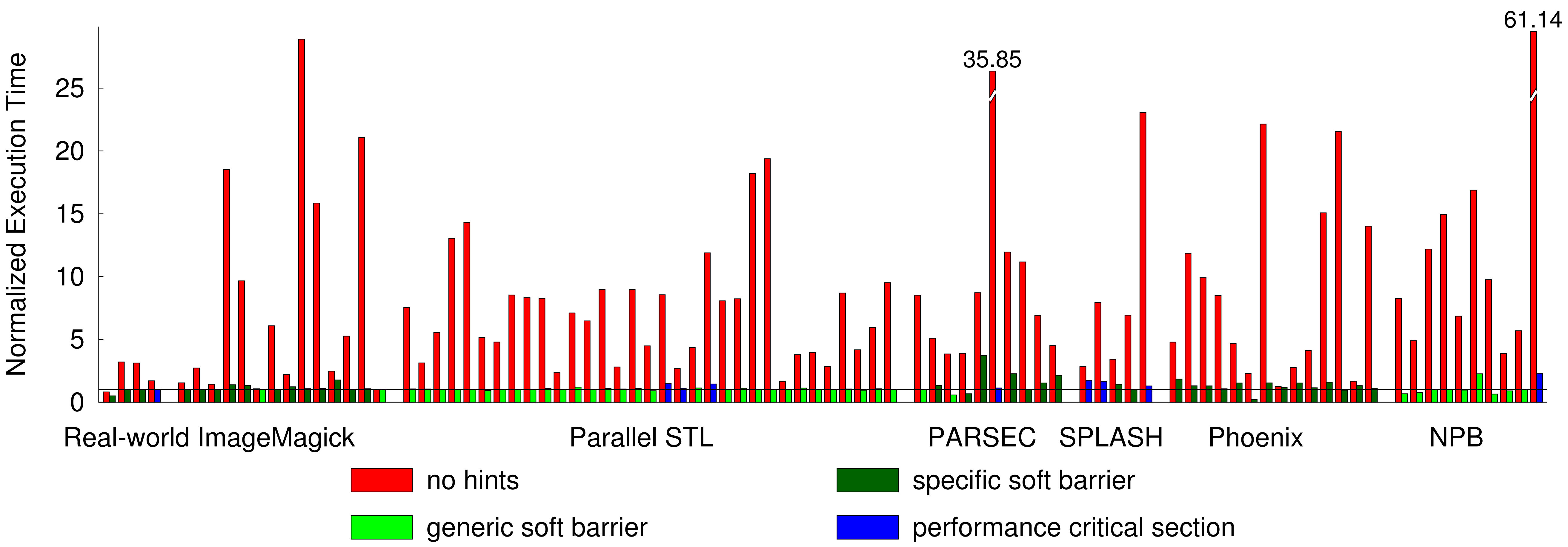
Comparison to Previous Work



Performance Overhead



Effect of Performance Hints

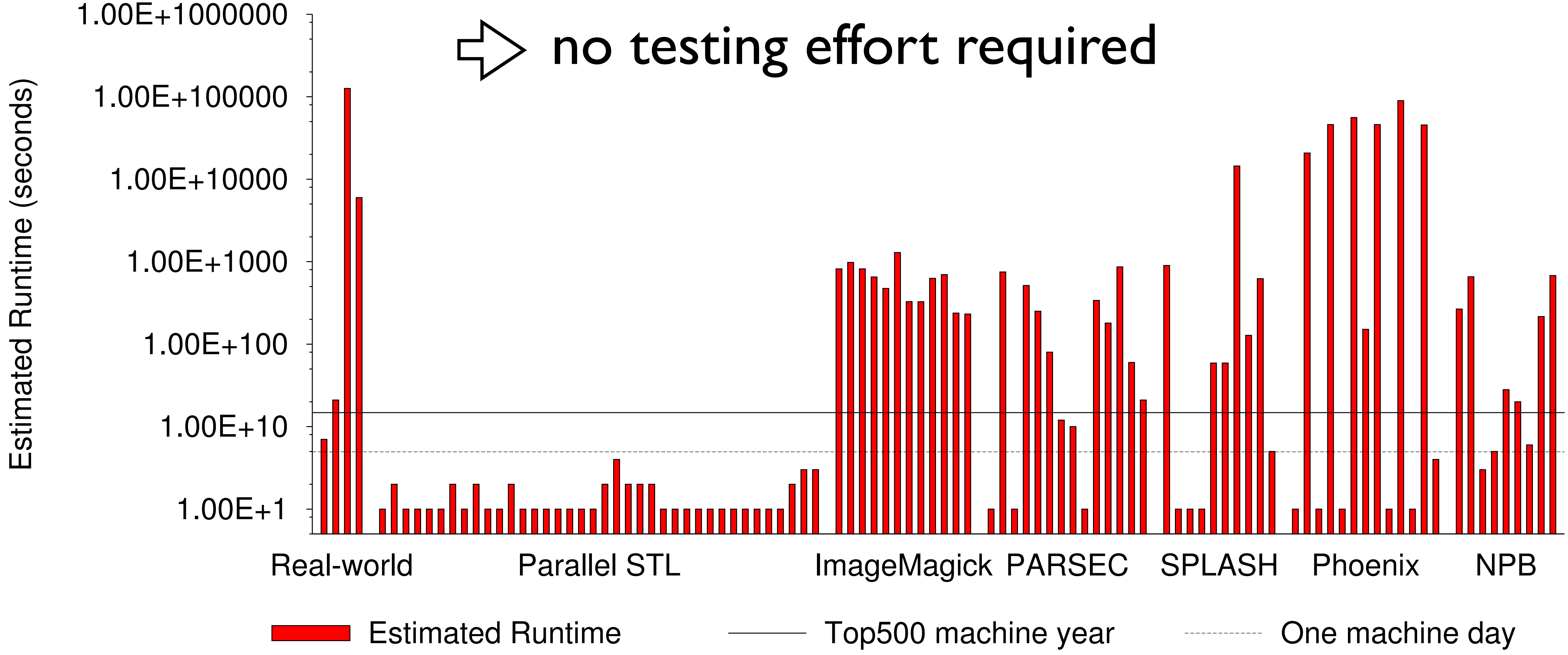


dBug without Parrot

dBug used to estimate testing effort

For these 96 programs Parrot uses single schedule

⇨ no testing effort required



Testing Nondeterminism

- For 12 programs Parrot uses more schedules
- 3 multiprocess programs, 9 performance CS

Program	dBug	dBug + Parrot
OpenLDAP	2.4E+2795	5.70E+1048
Redis	1.26E+08	9.11E+07
pfscan	2.43E+2117	32268
aget	2.05+17	5.11E+10
STL nth element	1.35E+07	8224
STL partial sort	1.37E+07	8194
STL partition	1.37E+07	8194
PARSEC fluidanimate	2.72E+218	2.64E+218
SPLASH cholesky	1.81E+371	5.99E+152
SPLASH fmm	1.25E+78	2.14E+54
SPLASH raytrace	1.08E+13863	3.68E+13755
NPB ua	N/A	N/A

Conclusion

- Parrot is a new practical thread runtime
  - By default synchronization events run deterministically
  - Programmers can use hints to tune performance
  - Improves testability without hurting performance
- Combining Parrot with dBug benefits both
  - dBug checks schedules that matter to Parrot
  - Parrot reduces # of schedules dBug needs to check

References

References

- Aviram[2010]: Amittai Aviram, Shu-Chun Weng, Sen Hu, and Bryan Ford, Efficient System-Enforced Deterministic Parallelism, OSDI 2010
- Bergan[2010]: Tom Bergan, Owen Anderson, Joseph Devietti, Luis Ceze, and Dan Grossman, CoreDet: A Compiler and Runtime System for Deterministic Multi-threaded Execution, ASPLOS 2010
- Berger[2009]: Emery Berger, Ting Yang, Tongping Liu, and Gene Novark, Grace: Safe Multithreaded Programming for C/C++, OOPSLA 2009
- Berger[2011]: Tongping Liu, Charlie Curtsinger, Emery D. Berger, dthreads: Efficient Deterministic Multi-threading, OSDI 2010

References

- Cui[2010]: Heming Cui, Jingyue Wu, Chia-Che Tsai, and Junfeng Yang, Stable Deterministic Multithreading through Schedule Memoization, OSDI 2010
- Cui[2011]: Heming Cui, Jingyue Wu, John Gallagher, Huayang Guo, Junfeng Yang, Efficient Deterministic Multithreading through Schedule Relaxation, SOSP 2011
- Devietti[2009]: Joseph Devietti, Brandon Lucia, Luis Ceze, and Mark Oskin, DMP: Deterministic Shared Memory Multiprocessing, ASPLOS 2009
- Olszewski[2009]: Marek Olszewski, Jason Ansel, and Saman Amarasinghe, Kendo: Efficient Deterministic Multi-threading in Software, ASPLOS 2009

References

- Simsa[2011]: Jiri Simsa, Garth Gibson, and Randy Bryant, dBug: Systematic Testing of Unmodified Distributed and Multi-threaded Systems, SSV 2011

Backup Slides

Contributing Students

- ISTC-CC students on this project
  - Jiri Simsa, Carnegie Mellon
  - Ben Blum, Carnegie Mellon
- Other advised students
  - Heming Cui, Columbia University
  - Yi-Hong Lin, Columbia University
  - Hao Li, Columbia University
  - Xinan Xu, Columbia University

Ease of Use: Soft Barriers

- 81 programs need soft barriers for performance
- 87 lines of hints in total

Program	Lines
mencoder, vips, swaptions, freqmine, facesim, x264, radiosity, radix, kmeans, linear-regression-pthread, linear-regression, matrix-multiply-pthread, matrix-multiply, word-count-pthread, string-match-pthread, string-match, histogram-pthread, histogram	2
PBZip2, ferret, kmeans-pthread, pca-pthread, pca, word-count	3
libgomp, bodytrack	4
ImageMagick (12 programs)	25

Ease of Use: Performance CS

- 9 programs need performance critical sections
- 22 lines of hints in total

Program	Lines	Nondet Sync Var
pfscan	2	matches_lock
partition	2	__result_lock
fluidanimate	6	mutex[i][j]
fmm	2	lock_array[i]
cholesky	2	tasks[i].taskLock
raytrace	2	ridlock
ua	6	tlock[i]

Parrot Scheduler

- Interface:
  - `void get_token(void);`
  - `void put_token(void);`
  - `int wait(void *addr, int timeout);`
  - `void signal(void *addr);`
  - `void broadcast(void *addr);`
  - `void nondet_begin(void);`
  - `void nondet_end(void);`
- Scheduling token is passed in a round-robin order
- Required for executing pthreads synchronizations
- Scheduler uses logical time (number of token passes)
- Non-deterministic regions:
  - Implement performance critical sections
  - Delimit network operations

Synchronization Wrappers

```
int pthread_mutex_lock_wrapper(pthread_mutex_t *mutex){
  scheduler.get_token();
  while(pthread_mutex_trylock(mutex)) {
    scheduler.wait(mutex, 0);
  }
  scheduler.put_token();
  return 0; // error handling omitted for clarity
}

int pthread_mutex_unlock_wrapper(pthread_mutex_t *mu){
  scheduler.get_token();
  pthread_mutex_unlock(mutex);
  scheduler.signal(mutex);
  scheduler.put_token();
  return 0; // error handling omitted for clarity
}
```