Tiled QR factorization algorithms

Henc Bouwmeester, Julien Langou
University of Colorado Denver
{Henricus.Bowmeester|Julien.Langou}@ucdenver.edu

Mathias Jacquelin[1] and Yves Robert[1,2]
1. Ecole Normale Supérieure de Lyon
2. ICL Knoxville
{Mathias.Jacquelin|Yves.Robert}@ens-lyon.fr

Paper available at http://arxiv.org/abs/1104.4475
Short version to appear in SC'2011

ICL Knoxville, June 17, 2011

Motivation

- Solving dense linear systems
- LU factorization not always stable enough
- QR factorization twice as costly but always robust
- Rectangular matrices (least squares)
- Use blocked algorithms to squeeze the most out of the hardware (BLAS3 level)
  $\Rightarrow$ **tile** operations instead of elementary operations

Motivation

- Solving dense linear systems
- LU factorization not always stable enough
- QR factorization twice as costly but always robust
- **Tall and skinny** rectangular matrices
- Use blocked algorithms to squeeze the most out of the hardware (BLAS3 level)
  $\Rightarrow$ **tile** operations instead of elementary operations

Outline

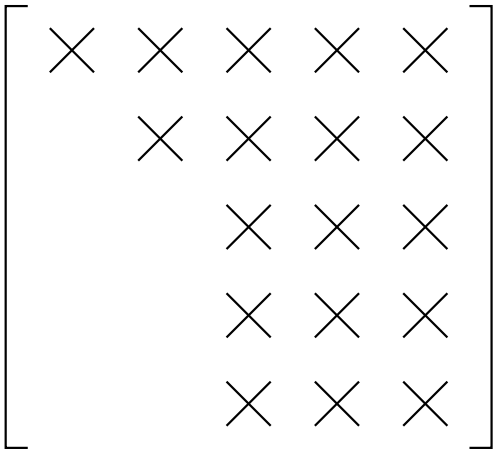
Outline

A naive QR algorithm

Algorithm 1: Naive QR algorithm for a tiled $p \times q$ matrix.

for k $= 1$ *to* $\min(p, q)$ **do**
  for i $= k + 1$ *to* p **do**
    $elim(i, piv(i, k), k)$

- k: panel index
- **orthogonal transformation** to zero out tile (i, k)

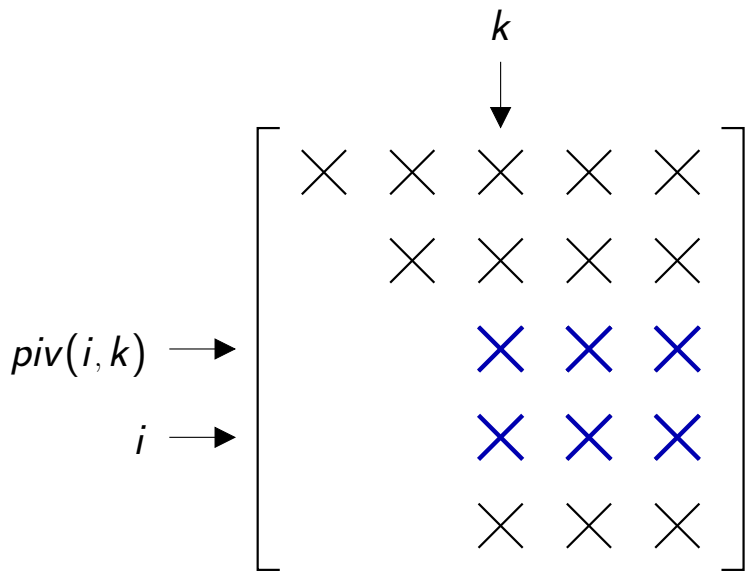


A naive QR algorithm

Algorithm 1: Naive QR algorithm for a tiled $p \times q$ matrix.

for k $= 1$ *to* $\min(p, q)$ **do**
  for i $= k + 1$ *to* p **do**
    $elim(i, piv(i, k), k)$

- k: panel index
- **orthogonal transformation** to zero out tile (i, k)

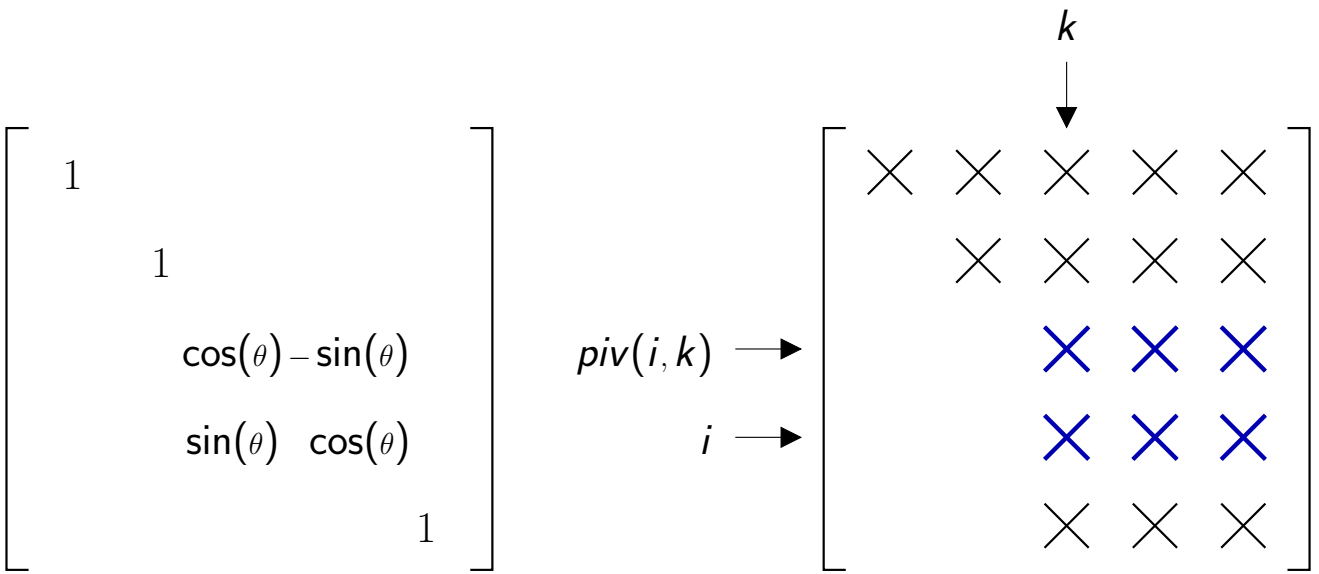


A naive QR algorithm

Algorithm 1: Naive QR algorithm for a tiled $p \times q$ matrix.

for k $= 1$ *to* $\min(p, q)$ **do**
  for i $= k + 1$ *to* p **do**
    $elim(i, piv(i, k), k)$

- k: panel index
- **orthogonal transformation** to zero out tile (i, k)

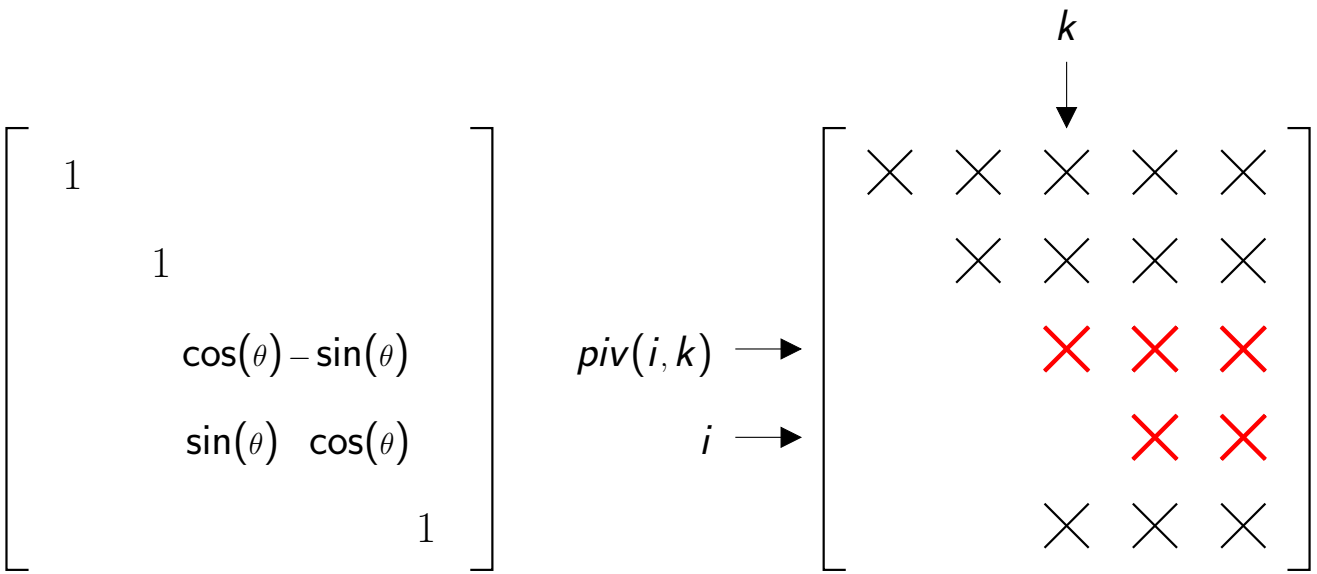


A naive QR algorithm

Algorithm 1: Naive QR algorithm for a tiled $p \times q$ matrix.

for k $= 1$ *to* $\min(p, q)$ **do**
  for i $= k + 1$ *to* p **do**
    $elim(i, piv(i, k), k)$

- k: panel index
- **orthogonal transformation** to zero out tile (i, k)

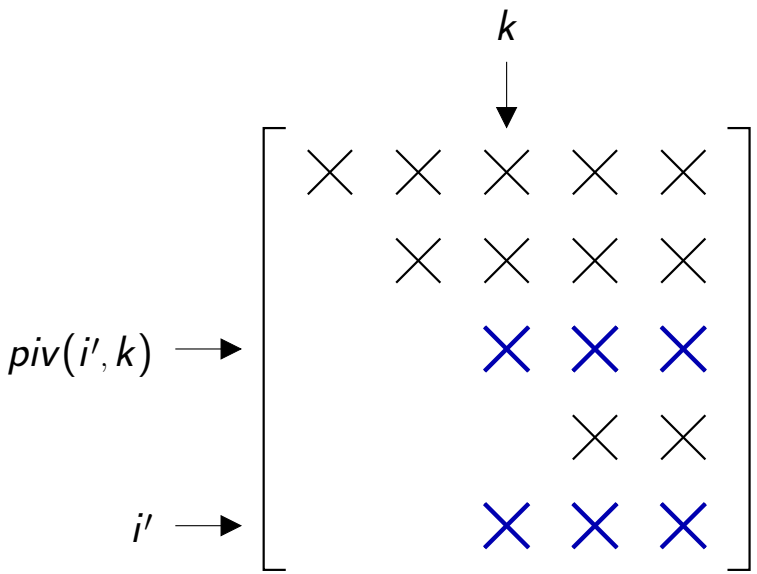


A naive QR algorithm

Algorithm 1: Naive QR algorithm for a tiled $p \times q$ matrix.

for k $= 1$ *to* $\min(p, q)$ **do**
  for i $= k + 1$ *to* p **do**
    $elim(i, piv(i, k), k)$

- k: panel index
- **orthogonal transformation** to zero out tile (i, k)

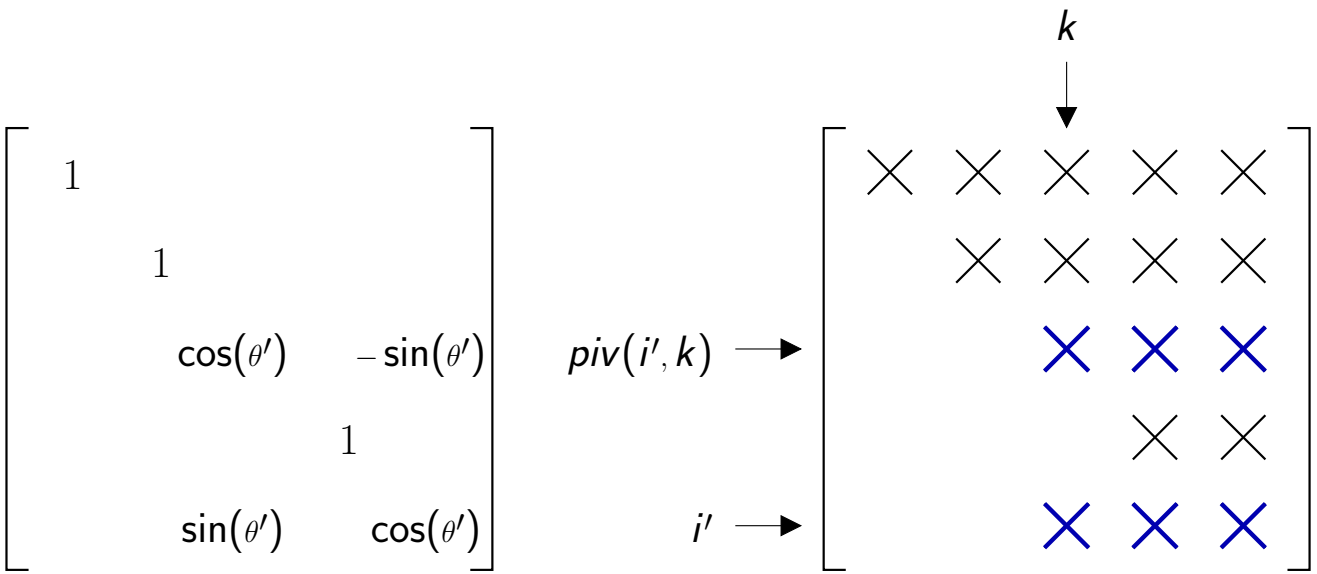


A naive QR algorithm

Algorithm 1: Naive QR algorithm for a tiled $p \times q$ matrix.

for k $= 1$ *to* $\min(p, q)$ **do**
  for i $= k + 1$ *to* p **do**
    $elim(i, piv(i, k), k)$

- k: panel index
- **orthogonal transformation** to zero out tile (i, k)

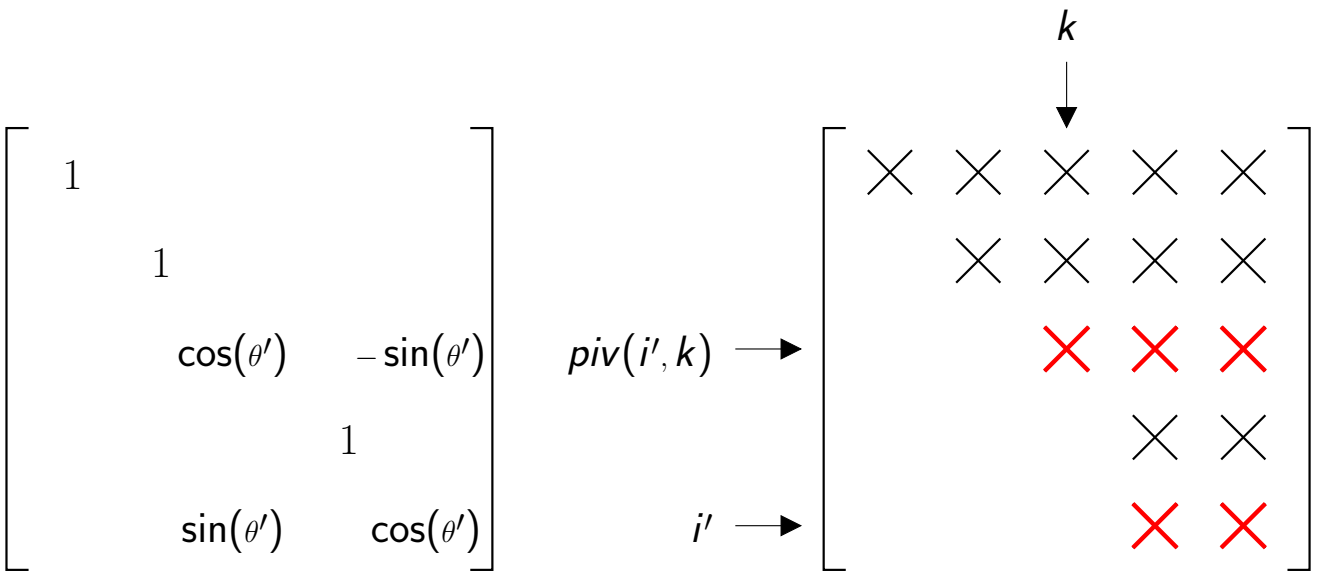


A naive QR algorithm

Algorithm 1: Naive QR algorithm for a tiled $p \times q$ matrix.

for k $= 1$ *to* $\min(p, q)$ **do**
  for i $= k + 1$ *to* p **do**
    $elim(i, piv(i, k), k)$

- k: panel index
- **orthogonal transformation** to zero out tile (i, k)



Kernels for orthogonal transformations

Operation	Panel		Update	
	Name	Cost	Name	Cost
Factor square into triangle	*GEQRT*	4	*UNMQR*	6
Zero square with triangle on top	*TSQRT*	6	*TSMQR*	12
Zero triangle with triangle on top	*TTQRT*	2	*TTMQR*	6

Algorithm 2: Elimination $elim(i, piv(i,k), k)$ via TS kernels.

$GEQRT(piv(i,k), k)$
for j $= k+1$ *to* q **do**
  $UNMQR(piv(i,k), k, j)$
$TSQRT(i, piv(i,k), k)$
for j $= k+1$ *to* q **do**
  $TSMQR(i, piv(i,k), k, j)$

Triangle on top of square

Algorithm 3: Elimination $elim(i, piv(i,k), k)$ via TT kernels.

$GEQRT(piv(i,k), k)$
$GEQRT(i,k)$
for j $= k+1$ *to* q **do**
  $UNMQR(piv(i,k), k, j)$
  $UNMQR(i,k,j)$
$TTQRT(i, piv(i,k), k)$
for j $= k+1$ *to* q **do**
  $TTMQR(i, piv(i,k), k, j)$

Triangle on top of triangle

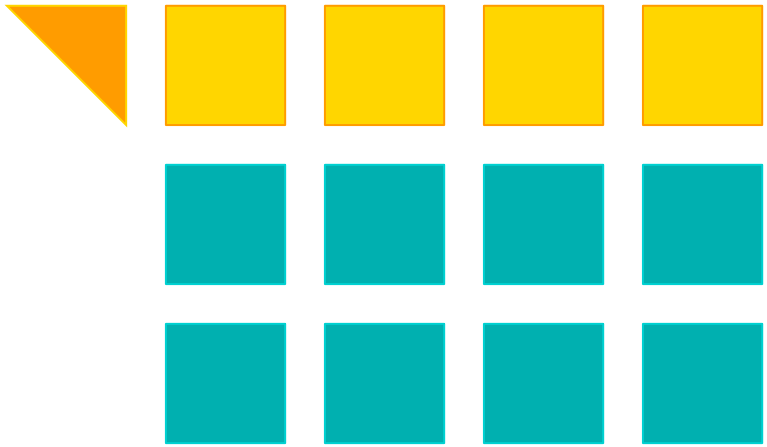
Kernels for orthogonal transformations

Operation	Panel		Update	
	Name	Cost	Name	Cost
Factor square into triangle	*GEQRT*	4	*UNMQR*	6
Zero square with triangle on top	*TSQRT*	6	*TSMQR*	12
Zero triangle with triangle on top	*TTQRT*	2	*TTMQR*	6

Algorithm 2: Elimination $elim(i, piv(i,k), k)$ via TS kernels.

$GEQRT(piv(i,k), k)$
for j $= k+1$ *to* q **do**
  $UNMQR(piv(i,k), k, j)$
$TSQRT(i, piv(i,k), k)$
for j $= k+1$ *to* q **do**
  $TSMQR(i, piv(i,k), k, j)$

Triangle on top of square



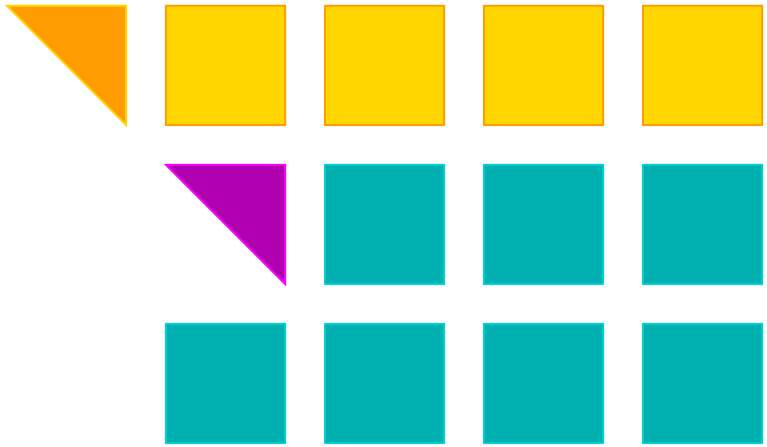
Kernels for orthogonal transformations

Operation	Panel		Update	
	Name	Cost	Name	Cost
Factor square into triangle	*GEQRT*	4	*UNMQR*	6
Zero square with triangle on top	*TSQRT*	6	*TSMQR*	12
Zero triangle with triangle on top	*TTQRT*	2	*TTMQR*	6

Algorithm 2: Elimination $elim(i, piv(i,k), k)$ via TS kernels.

$GEQRT(piv(i,k), k)$
for j $= k+1$ *to* q **do**
  $UNMQR(piv(i,k), k, j)$
$TSQRT(i, piv(i,k), k)$
for j $= k+1$ *to* q **do**
  $TSMQR(i, piv(i,k), k, j)$

Triangle on top of square



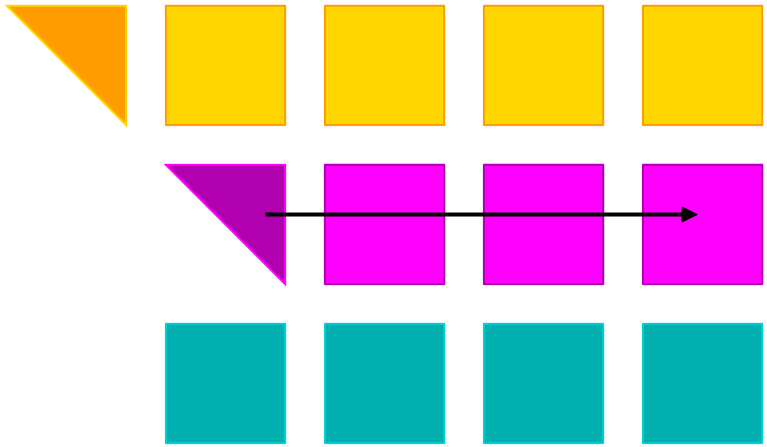
Kernels for orthogonal transformations

Operation	Panel		Update	
	Name	Cost	Name	Cost
Factor square into triangle	*GEQRT*	4	*UNMQR*	6
Zero square with triangle on top	*TSQRT*	6	*TSMQR*	12
Zero triangle with triangle on top	*TTQRT*	2	*TTMQR*	6

Algorithm 2: Elimination $elim(i, piv(i,k), k)$ via TS kernels.

$GEQRT(piv(i,k), k)$
for j $= k+1$ *to* q **do**
  $UNMQR(piv(i,k), k, j)$
$TSQRT(i, piv(i,k), k)$
for j $= k+1$ *to* q **do**
  $TSMQR(i, piv(i,k), k, j)$

Triangle on top of square



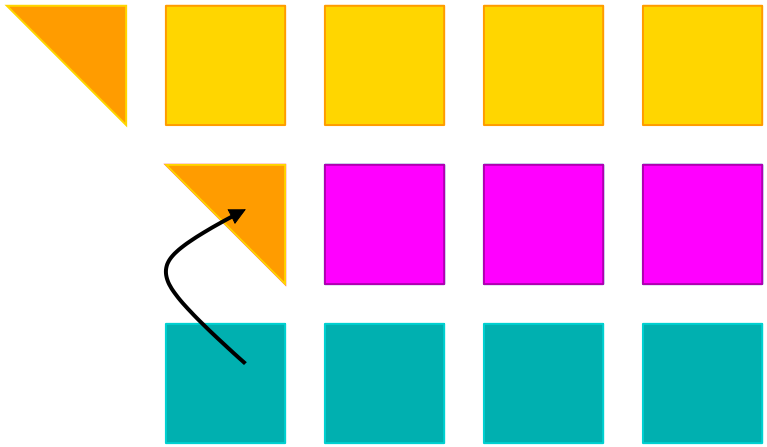
Kernels for orthogonal transformations

Operation	Panel		Update	
	Name	Cost	Name	Cost
Factor square into triangle	*GEQRT*	4	*UNMQR*	6
Zero square with triangle on top	*TSQRT*	6	*TSMQR*	12
Zero triangle with triangle on top	*TTQRT*	2	*TTMQR*	6

Algorithm 2: Elimination $elim(i, piv(i,k), k)$ via TS kernels.

$GEQRT(piv(i,k), k)$
for j $= k+1$ *to* q **do**
  $UNMQR(piv(i,k), k, j)$
$TSQRT(i, piv(i,k), k)$
for j $= k+1$ *to* q **do**
  $TSMQR(i, piv(i,k), k, j)$

Triangle on top of square



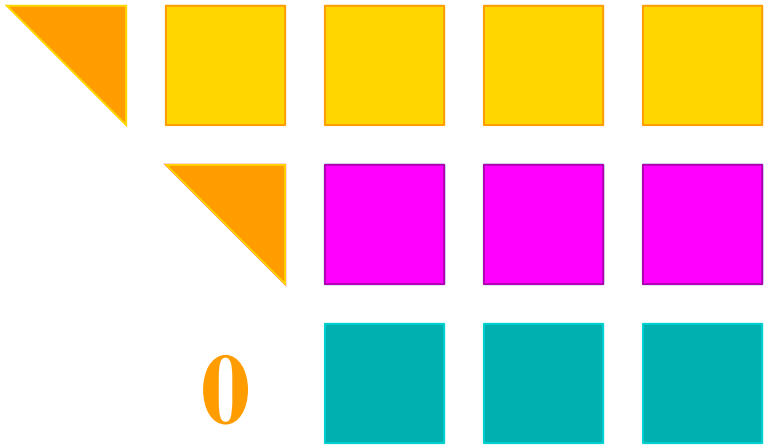
Kernels for orthogonal transformations

Operation	Panel		Update	
	Name	Cost	Name	Cost
Factor square into triangle	*GEQRT*	4	*UNMQR*	6
Zero square with triangle on top	*TSQRT*	6	*TSMQR*	12
Zero triangle with triangle on top	*TTQRT*	2	*TTMQR*	6

Algorithm 2: Elimination $elim(i, piv(i,k), k)$ via TS kernels.

$GEQRT(piv(i,k), k)$
for j $= k+1$ *to* q **do**
  $UNMQR(piv(i,k), k, j)$
$TSQRT(i, piv(i,k), k)$
for j $= k+1$ *to* q **do**
  $TSMQR(i, piv(i,k), k, j)$

Triangle on top of square



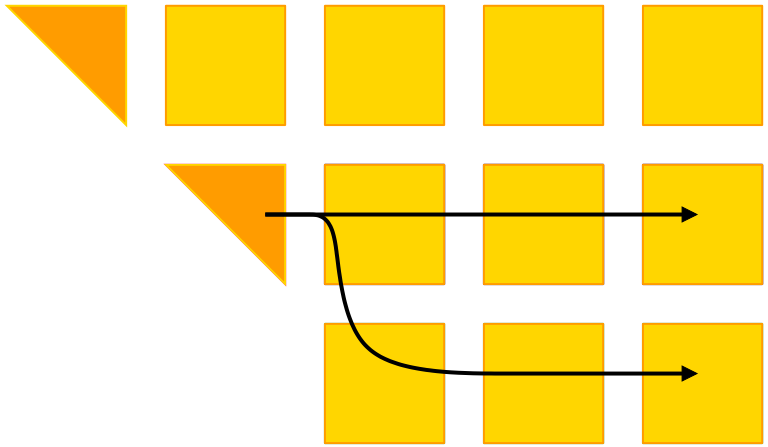
Kernels for orthogonal transformations

Operation	Panel		Update	
	Name	Cost	Name	Cost
Factor square into triangle	*GEQRT*	4	*UNMQR*	6
Zero square with triangle on top	*TSQRT*	6	*TSMQR*	12
Zero triangle with triangle on top	*TTQRT*	2	*TTMQR*	6

Algorithm 2: Elimination $elim(i, piv(i,k), k)$ via TS kernels.

$GEQRT(piv(i,k), k)$
for j $= k+1$ *to* q **do**
  $UNMQR(piv(i,k), k, j)$
$TSQRT(i, piv(i,k), k)$
for j $= k+1$ *to* q **do**
  $TSMQR(i, piv(i,k), k, j)$

Triangle on top of square



Kernels for orthogonal transformations

Operation	Panel		Update	
	Name	Cost	Name	Cost
Factor square into triangle	*GEQRT*	4	*UNMQR*	6
Zero square with triangle on top	*TSQRT*	6	*TSMQR*	12
Zero triangle with triangle on top	*TTQRT*	2	*TTMQR*	6

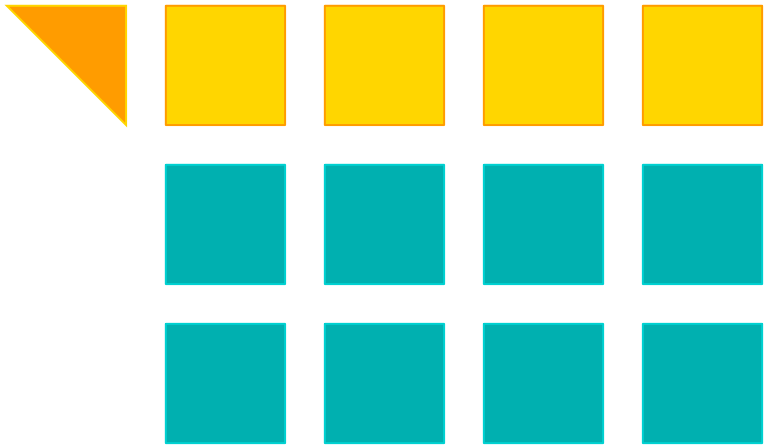
Algorithm 3: Elimination $elim(i, piv(i,k), k)$ via TT kernels.

$GEQRT(piv(i,k), k)$
$GEQRT(i,k)$
for j $= k+1$ *to* q **do**
  $UNMQR(piv(i,k), k, j)$
  $UNMQR(i,k,j)$
$TTQRT(i, piv(i,k), k)$
for j $= k+1$ *to* q **do**
  $TTMQR(i, piv(i,k), k, j)$

Triangle on top of triangle

Kernels for orthogonal transformations

Operation	Panel		Update	
	Name	Cost	Name	Cost
Factor square into triangle	*GEQRT*	4	*UNMQR*	6
Zero square with triangle on top	*TSQRT*	6	*TSMQR*	12
Zero triangle with triangle on top	*TTQRT*	2	*TTMQR*	6



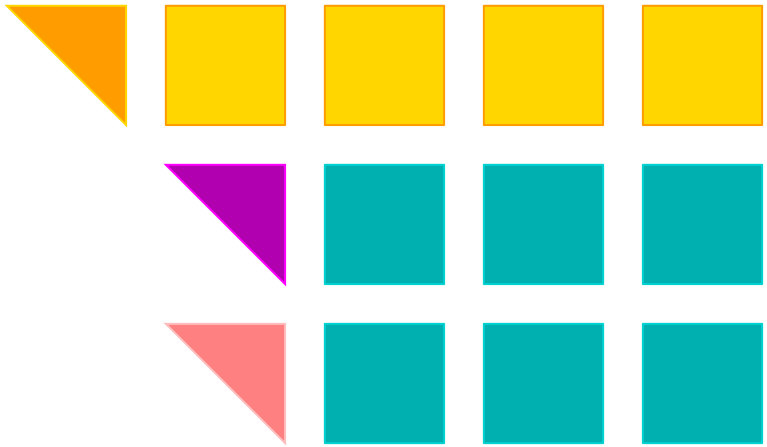
Algorithm 3: Elimination $elim(i, piv(i,k), k)$ via TT kernels.

$GEQRT(piv(i,k), k)$
$GEQRT(i,k)$
for j $= k+1$ *to* q **do**
  $UNMQR(piv(i,k), k, j)$
  $UNMQR(i,k,j)$
$TTQRT(i, piv(i,k), k)$
for j $= k+1$ *to* q **do**
  $TTMQR(i, piv(i,k), k, j)$

Triangle on top of triangle

Kernels for orthogonal transformations

Operation	Panel		Update	
	Name	Cost	Name	Cost
Factor square into triangle	*GEQRT*	4	*UNMQR*	6
Zero square with triangle on top	*TSQRT*	6	*TSMQR*	12
Zero triangle with triangle on top	*TTQRT*	2	*TTMQR*	6



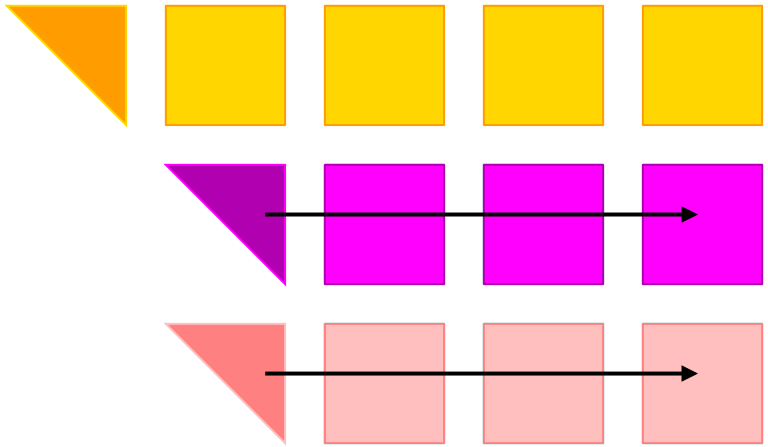
Algorithm 3: Elimination $elim(i, piv(i,k), k)$ via TT kernels.

$GEQRT(piv(i,k), k)$
$GEQRT(i,k)$
for j $= k+1$ *to* q **do**
  $UNMQR(piv(i,k), k, j)$
  $UNMQR(i,k,j)$
$TTQRT(i, piv(i,k), k)$
for j $= k+1$ *to* q **do**
  $TTMQR(i, piv(i,k), k, j)$

Triangle on top of triangle

Kernels for orthogonal transformations

Operation	Panel		Update	
	Name	Cost	Name	Cost
Factor square into triangle	*GEQRT*	4	*UNMQR*	6
Zero square with triangle on top	*TSQRT*	6	*TSMQR*	12
Zero triangle with triangle on top	*TTQRT*	2	*TTMQR*	6



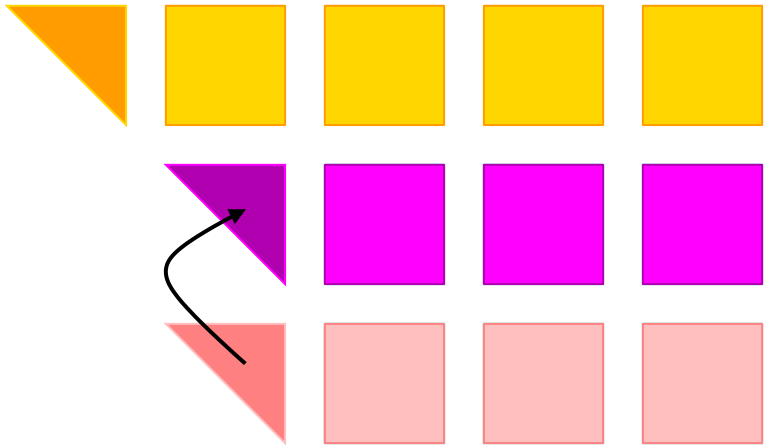
Algorithm 3: Elimination $elim(i, piv(i,k), k)$ via TT kernels.

$GEQRT(piv(i,k), k)$
$GEQRT(i,k)$
for j $= k+1$ *to* q **do**
  $UNMQR(piv(i,k), k, j)$
  $UNMQR(i,k,j)$
$TTQRT(i, piv(i,k), k)$
for j $= k+1$ *to* q **do**
  $TTMQR(i, piv(i,k), k, j)$

Triangle on top of triangle

Kernels for orthogonal transformations

Operation	Panel		Update	
	Name	Cost	Name	Cost
Factor square into triangle	*GEQRT*	4	*UNMQR*	6
Zero square with triangle on top	*TSQRT*	6	*TSMQR*	12
Zero triangle with triangle on top	*TTQRT*	2	*TTMQR*	6



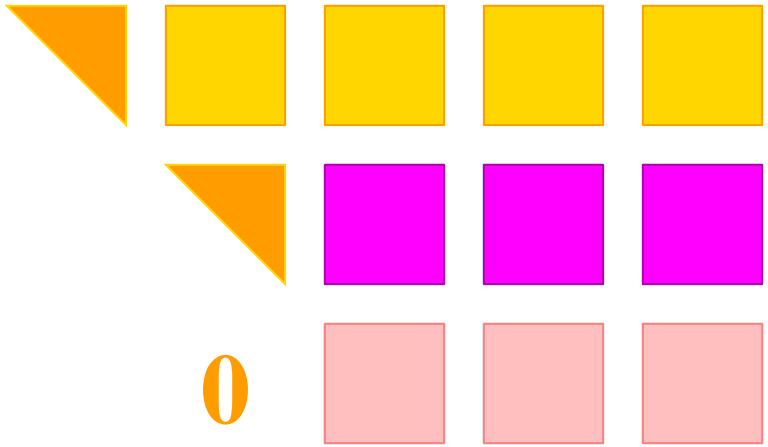
Algorithm 3: Elimination $elim(i, piv(i,k), k)$ via TT kernels.

$GEQRT(piv(i,k), k)$
$GEQRT(i,k)$
for j $= k+1$ *to* q **do**
  $UNMQR(piv(i,k), k, j)$
  $UNMQR(i,k,j)$
$TTQRT(i, piv(i,k), k)$
for j $= k+1$ *to* q **do**
  $TTMQR(i, piv(i,k), k, j)$

Triangle on top of triangle

Kernels for orthogonal transformations

Operation	Panel		Update	
	Name	Cost	Name	Cost
Factor square into triangle	*GEQRT*	4	*UNMQR*	6
Zero square with triangle on top	*TSQRT*	6	*TSMQR*	12
Zero triangle with triangle on top	*TTQRT*	2	*TTMQR*	6



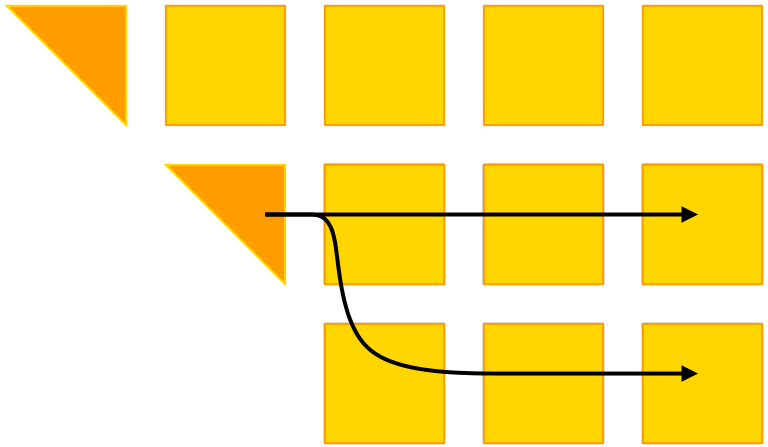
Algorithm 3: Elimination $elim(i, piv(i,k), k)$ via TT kernels.

$GEQRT(piv(i,k), k)$
$GEQRT(i,k)$
for j $= k+1$ *to* q **do**
  $UNMQR(piv(i,k), k, j)$
  $UNMQR(i,k,j)$
$TTQRT(i, piv(i,k), k)$
for j $= k+1$ *to* q **do**
  $TTMQR(i, piv(i,k), k, j)$

Triangle on top of triangle

Kernels for orthogonal transformations

Operation	Panel		Update	
	Name	Cost	Name	Cost
Factor square into triangle	*GEQRT*	4	*UNMQR*	6
Zero square with triangle on top	*TSQRT*	6	*TSMQR*	12
Zero triangle with triangle on top	*TTQRT*	2	*TTMQR*	6



Algorithm 3: Elimination $elim(i, piv(i,k), k)$ via TT kernels.

$GEQRT(piv(i,k), k)$
$GEQRT(i,k)$
for j $= k+1$ *to* q **do**
  $UNMQR(piv(i,k), k, j)$
  $UNMQR(i,k,j)$
$TTQRT(i, piv(i,k), k)$
for j $= k+1$ *to* q **do**
  $TTMQR(i, piv(i,k), k, j)$

Triangle on top of triangle

Kernels for orthogonal transformations

Operation	Panel		Update	
	Name	Cost	Name	Cost
Factor square into triangle	*GEQRT*	4	*UNMQR*	6
Zero square with triangle on top	*TSQRT*	6	*TSMQR*	12
Zero triangle with triangle on top	*TTQRT*	2	*TTMQR*	6

Algorithm 2: Elimination $elim(i, piv(i,k), k)$ via TS kernels.

$GEQRT(piv(i,k), k)$
for j $= k+1$ *to* q **do**
  $UNMQR(piv(i,k), k, j)$
$TSQRT(i, piv(i,k), k)$
for j $= k+1$ *to* q **do**
  $TSMQR(i, piv(i,k), k, j)$

Triangle on top of square

Algorithm 3: Elimination $elim(i, piv(i,k), k)$ via TT kernels.

$GEQRT(piv(i,k), k)$
$GEQRT(i,k)$
for j $= k+1$ *to* q **do**
  $UNMQR(piv(i,k), k, j)$
  $UNMQR(i,k,j)$
$TTQRT(i, piv(i,k), k)$
for j $= k+1$ *to* q **do**
  $TTMQR(i, piv(i,k), k, j)$

Triangle on top of triangle

Kernels

- *Triangle on top of triangle*
  $\rightarrow$ more parallelism ☺
- *Triangle on top of square*
  $\rightarrow$ more data locality
  $\rightarrow$ efficient PLASMA implementation

Approach

- Design new TT algorithms
- Compare with existing TS algorithms

Execution schemes

Algorithm = *elimination list*

$$\begin{matrix} x & x & x & \dots \\ x & x & x & \dots \\ x & x & x & \dots \\ x & x & x & \dots \\ x & x & x & \dots \\ x & x & x & \dots \end{matrix}$$

Execution schemes

Algorithm = *elimination list*

$elim(3, 1, 1)$

$$\begin{matrix} x & x & x & \dots \\ x & x & x & \dots \\ 1 & x & x & \dots \\ x & x & x & \dots \\ x & x & x & \dots \\ x & x & x & \dots \end{matrix}$$

Execution schemes

Algorithm = *elimination list*

$elim(3, 1, 1)$ $elim(6, 4, 1)$

$$\begin{matrix} x & x & x & \dots \\ x & x & x & \dots \\ 1 & x & x & \dots \\ x & x & x & \dots \\ x & x & x & \dots \\ 1 & x & x & \dots \end{matrix}$$

Execution schemes

Algorithm = *elimination list*

$elim(3,1,1)$ $elim(6,4,1)$ $elim(2,1,1)$

$$\begin{matrix} x & x & x & \dots \\ 2 & x & x & \dots \\ 1 & x & x & \dots \\ x & x & x & \dots \\ x & x & x & \dots \\ 1 & x & x & \dots \end{matrix}$$

Execution schemes

Algorithm = *elimination list*

$elim(3,1,1)$ $elim(6,4,1)$ $elim(2,1,1)$
$elim(5,4,1)$

$$\begin{matrix} x & x & x & \dots \\ 2 & x & x & \dots \\ 1 & x & x & \dots \\ x & x & x & \dots \\ 2 & x & x & \dots \\ 1 & x & x & \dots \end{matrix}$$

Execution schemes

Algorithm = *elimination list*

$elim(3,1,1)$ $elim(6,4,1)$ $elim(2,1,1)$
$elim(5,4,1)$ $elim(4,1,1)$

$$\begin{matrix} x & x & x & \dots \\ 2 & x & x & \dots \\ 1 & x & x & \dots \\ 3 & x & x & \dots \\ 2 & x & x & \dots \\ 1 & x & x & \dots \end{matrix}$$

Execution schemes

Algorithm = *elimination list*

$elim(3,1,1)$ $elim(6,4,1)$ $elim(2,1,1)$
$elim(5,4,1)$ $elim(4,1,1)$ $elim(6,5,2)$

$$\begin{matrix} x & x & x & \dots \\ 2 & x & x & \dots \\ 1 & x & x & \dots \\ 3 & x & x & \dots \\ 2 & x & x & \dots \\ 1 & 3 & x & \dots \end{matrix}$$

Execution schemes

Algorithm = *elimination list*

$elim(3,1,1)$ $elim(6,4,1)$ $elim(2,1,1)$
$elim(5,4,1)$ $elim(4,1,1)$ $elim(6,5,2)$ $elim(3,2,2)$

$$\begin{array}{cccc} x & x & x & \dots \\ 2 & x & x & \dots \\ 1 & 3 & x & \dots \\ 3 & x & x & \dots \\ 2 & x & x & \dots \\ 1 & 3 & x & \dots \end{array}$$

Execution schemes

Algorithm = *elimination list*

$elim(3,1,1)$ $elim(6,4,1)$ $elim(2,1,1)$
$elim(5,4,1)$ $elim(4,1,1)$ $elim(6,5,2)$ $elim(3,2,2)$

$$\begin{array}{cccc} x & x & x & \dots \\ 2 & x & x & \dots \\ 1 & 3 & x & \dots \\ 3 & x & x & \dots \\ 2 & x & x & \dots \\ 1 & 3 & x & \dots \end{array}$$

Factor operations trigger update kernels
ASAP execution of elimination list!!

Elimination list

- Eliminate all tiles once and only once:
  $\Rightarrow$ for $i > k$, exactly one entry $elim(i, \star, k)$
  ☺ All such schemes perform same amount of computations

- Validity of $elim(i, piv(i, k), k)$:
  - Both rows i and $piv(i, k)$ ready
  - Row $piv(i, k)$ potential annihilator

Elimination list

- Eliminate all tiles once and only once:
  $\Rightarrow$ for $i > k$, exactly one entry $elim(i, \star, k)$
  ☺ All such schemes perform same amount of computations

- Validity of $elim(i, piv(i, k), k)$:
  - Both rows i and $piv(i, k)$ ready
    $\rightarrow$ tiles left of panel already zeroed out

  - Row $piv(i, k)$ potential annihilator
    $\rightarrow$ tile $(piv(i, k), k)$ not zeroed out yet

Elimination list

- Eliminate all tiles once and only once:
  $\Rightarrow$ for $i > k$, exactly one entry $elim(i, \star, k)$
  ☺ All such schemes perform same amount of computations

- Validity of $elim(i, piv(i,k), k)$:
  - Both rows i and $piv(i,k)$ ready
    $\rightarrow$ tiles left of panel already zeroed out
    $\rightarrow$ $\begin{cases} elim(i, piv(i,k'), k') & \text{are predecessors} \\ elim(piv(i,k), piv(piv(i,k),k'), k') & (k' < k) \end{cases}$
  - Row $piv(i,k)$ potential annihilator
    $\rightarrow$ tile $(piv(i,k), k)$ not zeroed out yet
    $\rightarrow$ $elim(piv(i,k), piv(piv(i,k),k), k)$ is a successor

Elimination list

- Eliminate all tiles once and only once:
  $\Rightarrow$ for $i > k$, exactly one entry $elim(i, \star, k)$
  ☺ All such schemes perform same amount of computations

- Validity of $elim(i, piv(i, k), k)$:
  - Both rows i and $piv(i, k)$ ready
    $\rightarrow$ tiles left of panel already zeroed out
    $\rightarrow$ $\begin{cases} elim(i, piv(i, k'), k') & \text{are predecessors} \\ elim(piv(i, k), piv(piv(i, k), k'), k') & (k' < k) \end{cases}$
  - Row $piv(i, k)$ potential annihilator
    $\rightarrow$ tile $(piv(i, k), k)$ not zeroed out yet
    $\rightarrow$ $elim(piv(i, k), piv(piv(i, k), k), k)$ is a successor

$\Rightarrow$ **Generic tiled algorithm**

Dominance of bottom-first algorithms

Lemma

Any generic tiled algorithm can be modified, without changing its execution time, so that all eliminations $elim(i, piv(i,k), k)$ *satisfy to*

$$i > piv(i,k)$$

Outline

Outline

Critical path of coarse-grain algorithms

- Time-unit: orthogonal transformation across two rows
- Inspired by algorithms introduced twenty-five years ago
  - SAMEH-KUCK: panel row for all eliminations in a column
  - FIBONACCI: Fibonacci scheme of order 1 (Modi and Clarke)
  - GREEDY: eliminates as many tiles as possible in each column, starting with bottom rows

(a) SAMEH-KUCK					
⋆					
1	⋆				
2	3	⋆			
3	4	5	⋆		
4	5	6	7	⋆	
5	6	7	8	9	⋆
6	7	8	9	10	11
7	8	9	10	11	12
8	9	10	11	12	13
9	10	11	12	13	14
10	11	12	13	14	15
11	12	13	14	15	16
12	13	14	15	16	17
13	14	15	16	17	18
14	15	16	17	18	19

(b) FIBONACCI					
⋆					
5	⋆				
4	7	⋆			
4	6	9	⋆		
3	6	8	11	⋆	
3	5	8	10	13	⋆
3	5	7	10	12	15
2	5	7	9	12	14
2	4	7	9	11	14
2	4	6	9	11	13
2	4	6	8	11	13
1	4	6	8	10	13
1	3	6	8	10	12
1	3	5	8	10	12
1	3	5	7	10	12

(c) GREEDY					
⋆					
4	⋆				
3	6	⋆			
3	5	8	⋆		
2	5	7	10	⋆	
2	4	7	9	12	⋆
2	4	6	9	11	14
2	4	6	8	10	13
1	3	5	8	10	12
1	3	5	7	9	11
1	3	5	7	9	11
1	3	4	6	8	10
1	2	4	6	8	10
1	2	4	5	7	9
1	2	3	5	6	8

Critical path of coarse-grain algorithms

(a) SAMEH-KUCK

⋆					
1	⋆				
2	3	⋆			
3	4	5	⋆		
4	5	6	7	⋆	
5	6	7	8	9	⋆
6	7	8	9	10	11
7	8	9	10	11	12
8	9	10	11	12	13
9	10	11	12	13	14
10	11	12	13	14	15
11	12	13	14	15	16
12	13	14	15	16	17
13	14	15	16	17	18
14	15	16	17	18	19

Critical path of coarse-grain algorithms

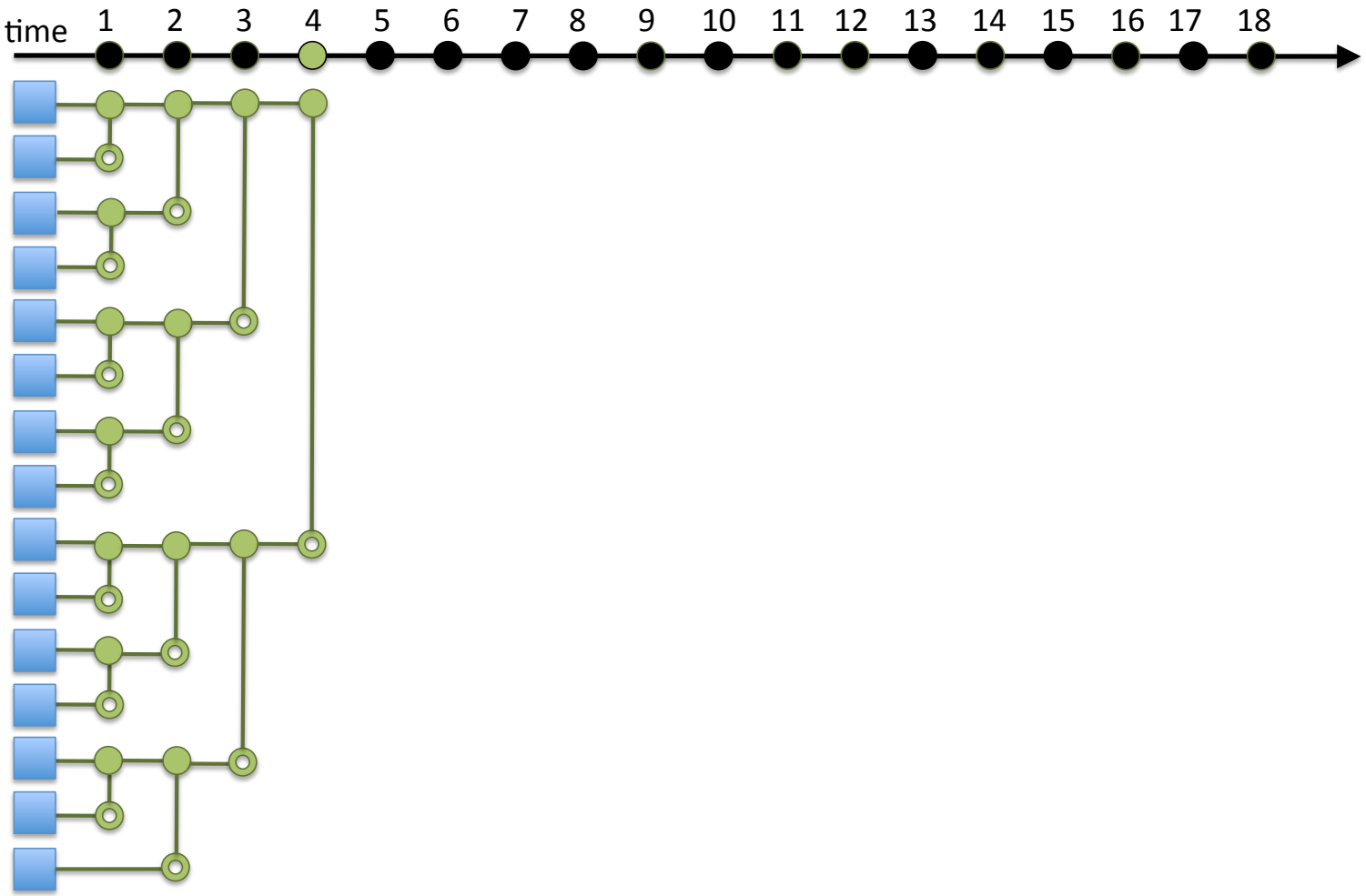
(b) FIBONACCI					
$\star$					
5	$\star$				
4	7	$\star$			
4	6	9	$\star$		
3	6	8	11	$\star$	
3	5	8	10	13	$\star$
3	5	7	10	12	15
2	5	7	9	12	14
2	4	7	9	11	14
2	4	6	9	11	13
2	4	6	8	11	13
1	4	6	8	10	13
1	3	6	8	10	12
1	3	5	8	10	12
1	3	5	7	10	12

Critical path of coarse-grain algorithms

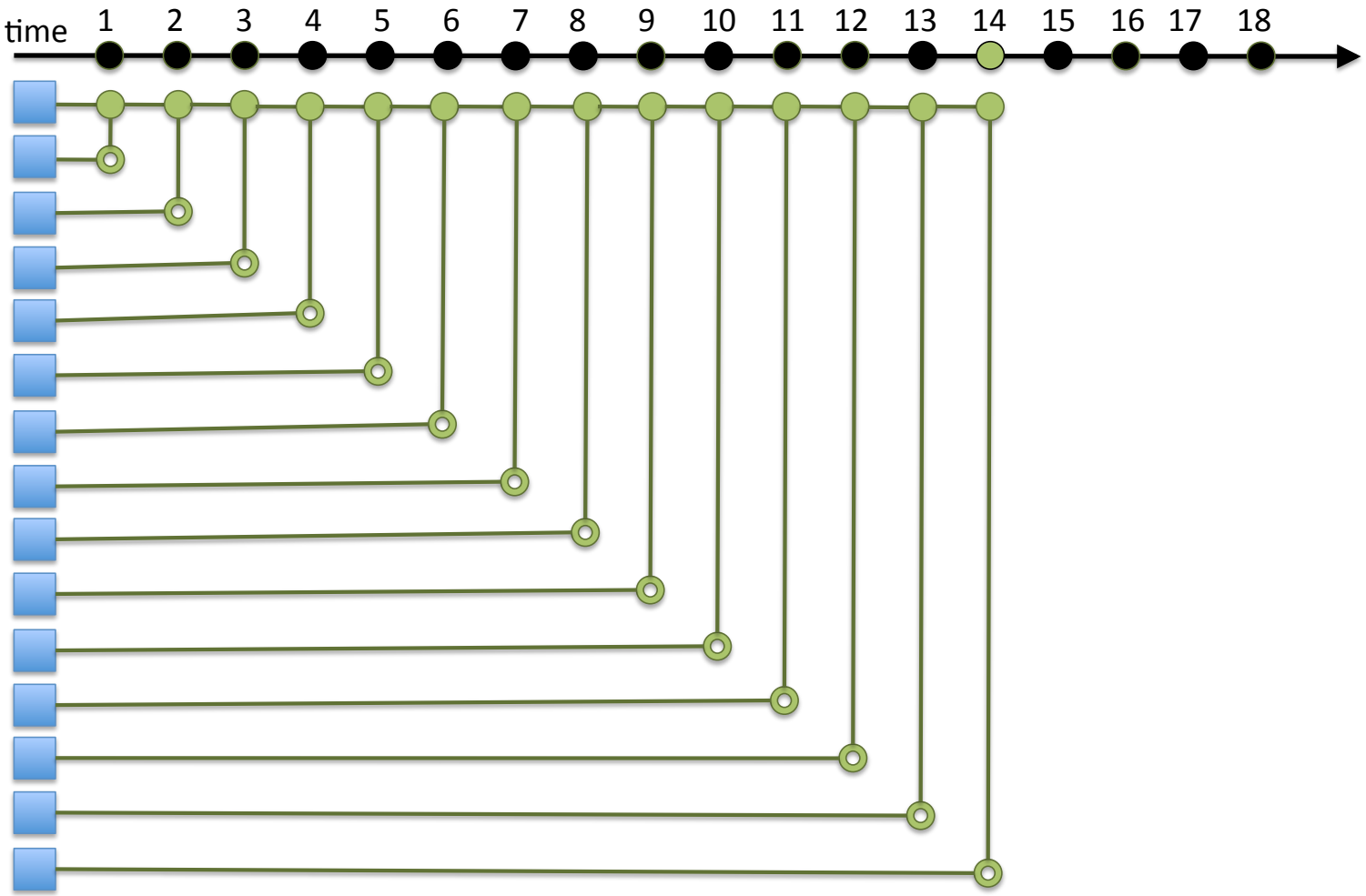
(c) GREEDY					
$\star$					
4	$\star$				
3	6	$\star$			
3	5	8	$\star$		
2	5	7	10	$\star$	
2	4	7	9	12	$\star$
2	4	6	9	11	14
2	4	6	8	10	13
1	3	5	8	10	12
1	3	5	7	9	11
1	3	5	7	9	11
1	3	4	6	8	10
1	2	4	6	8	10
1	2	4	5	7	9
1	2	3	5	6	8

How well do you pipeline?

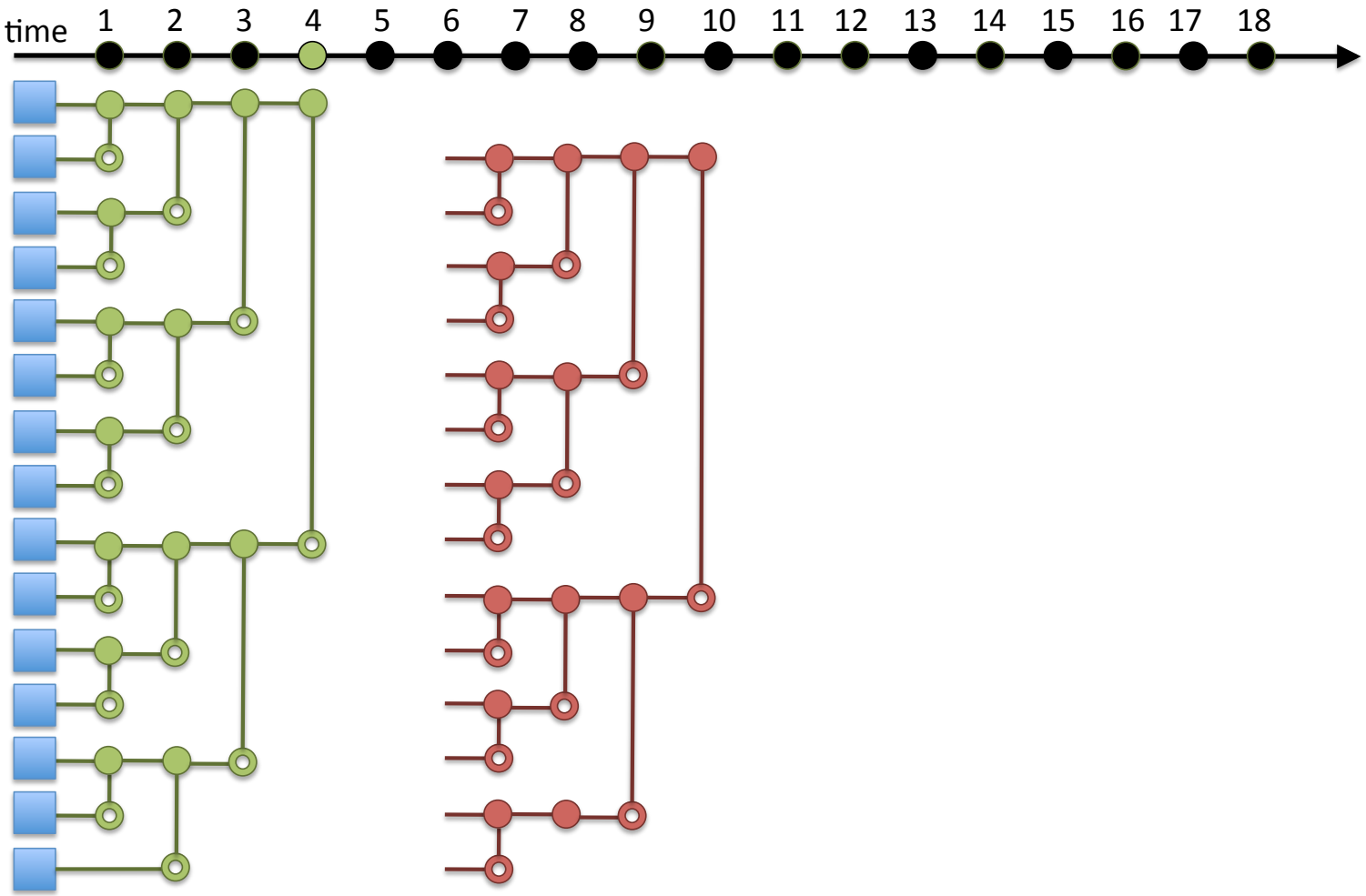
How well do you pipeline? BinaryTree



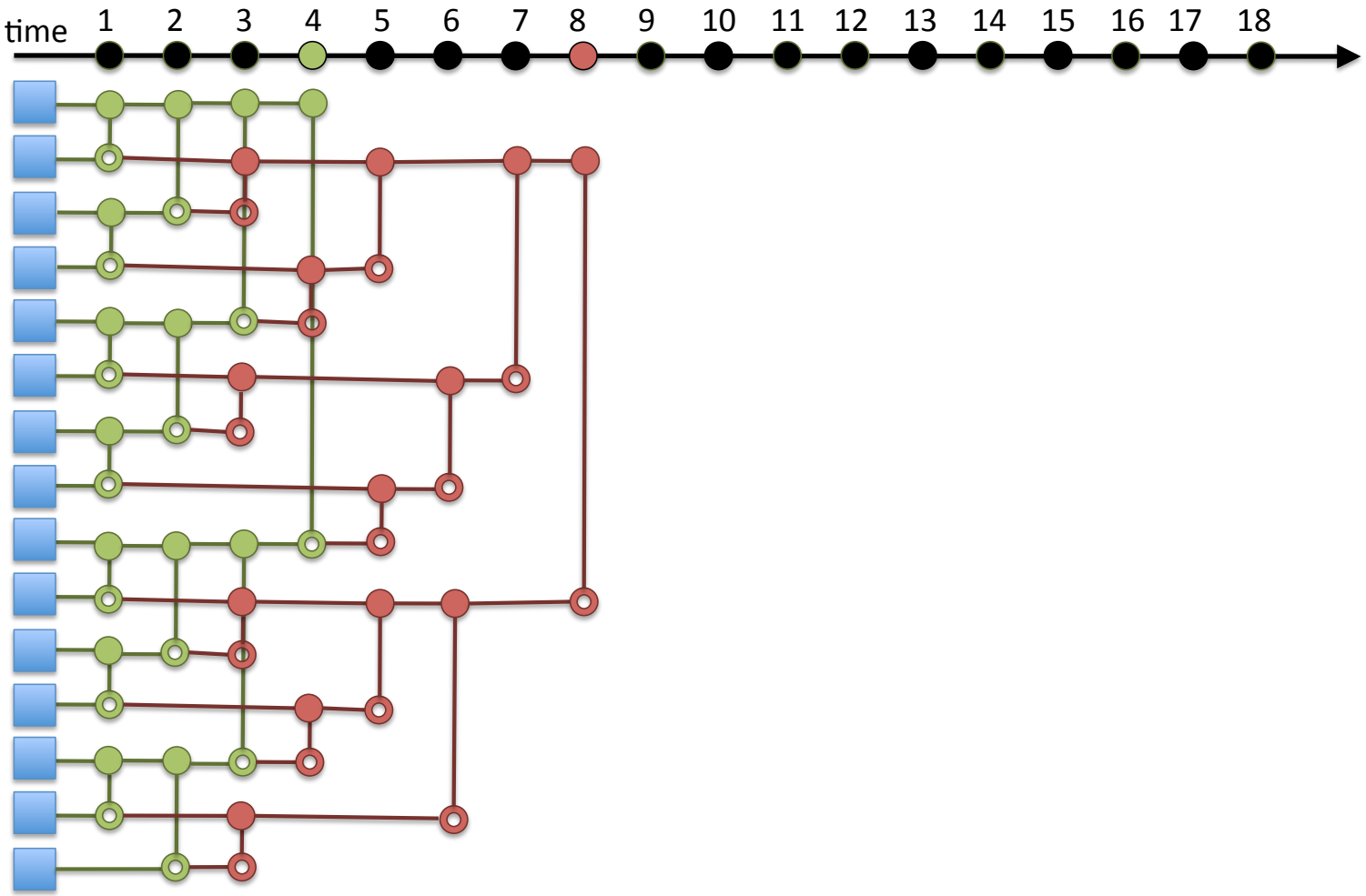
How well do you pipeline? FlatTree



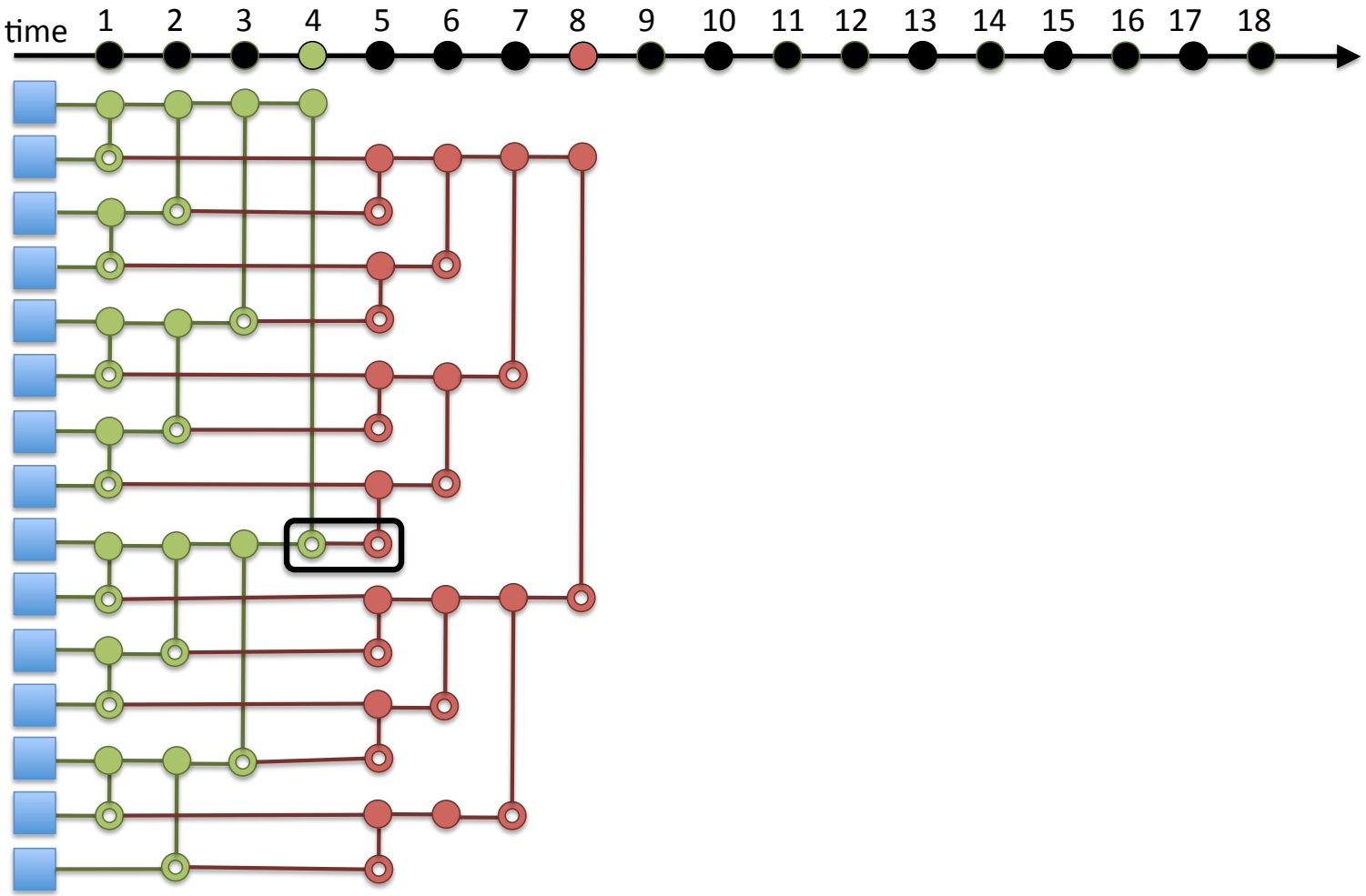
How well do you pipeline? BinaryTree



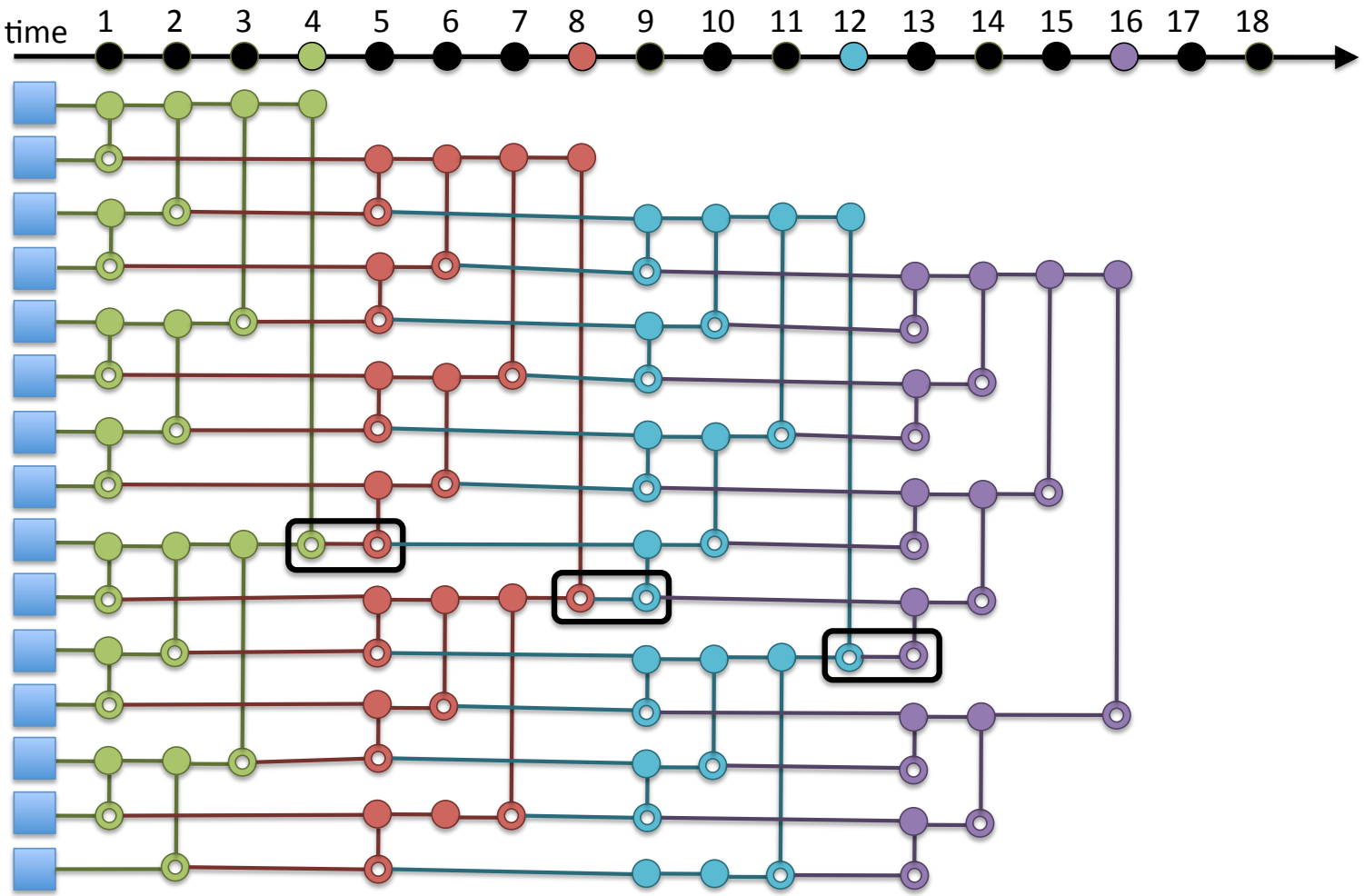
How well do you pipeline? BinaryTree



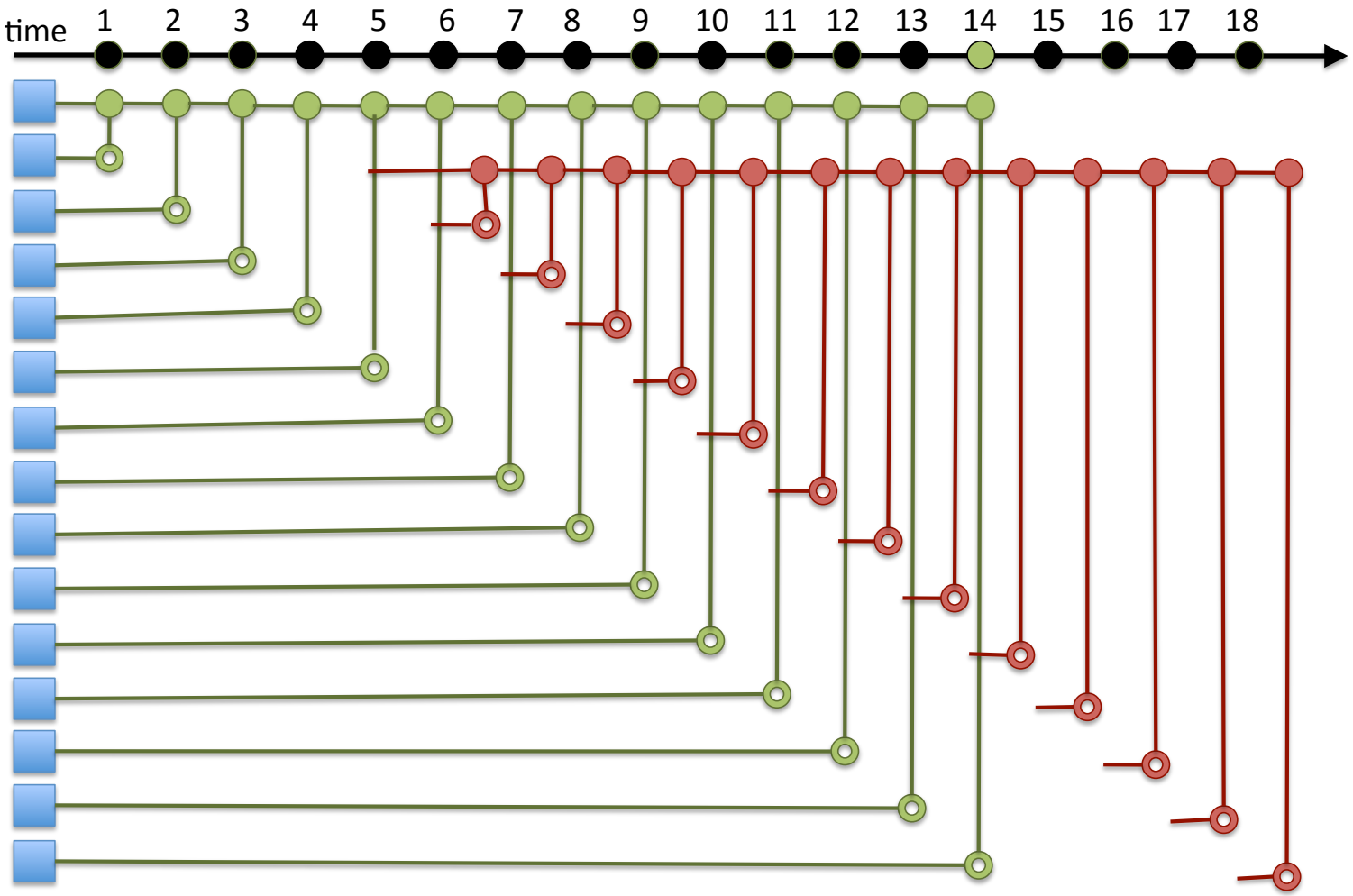
How well do you pipeline? BinaryTree



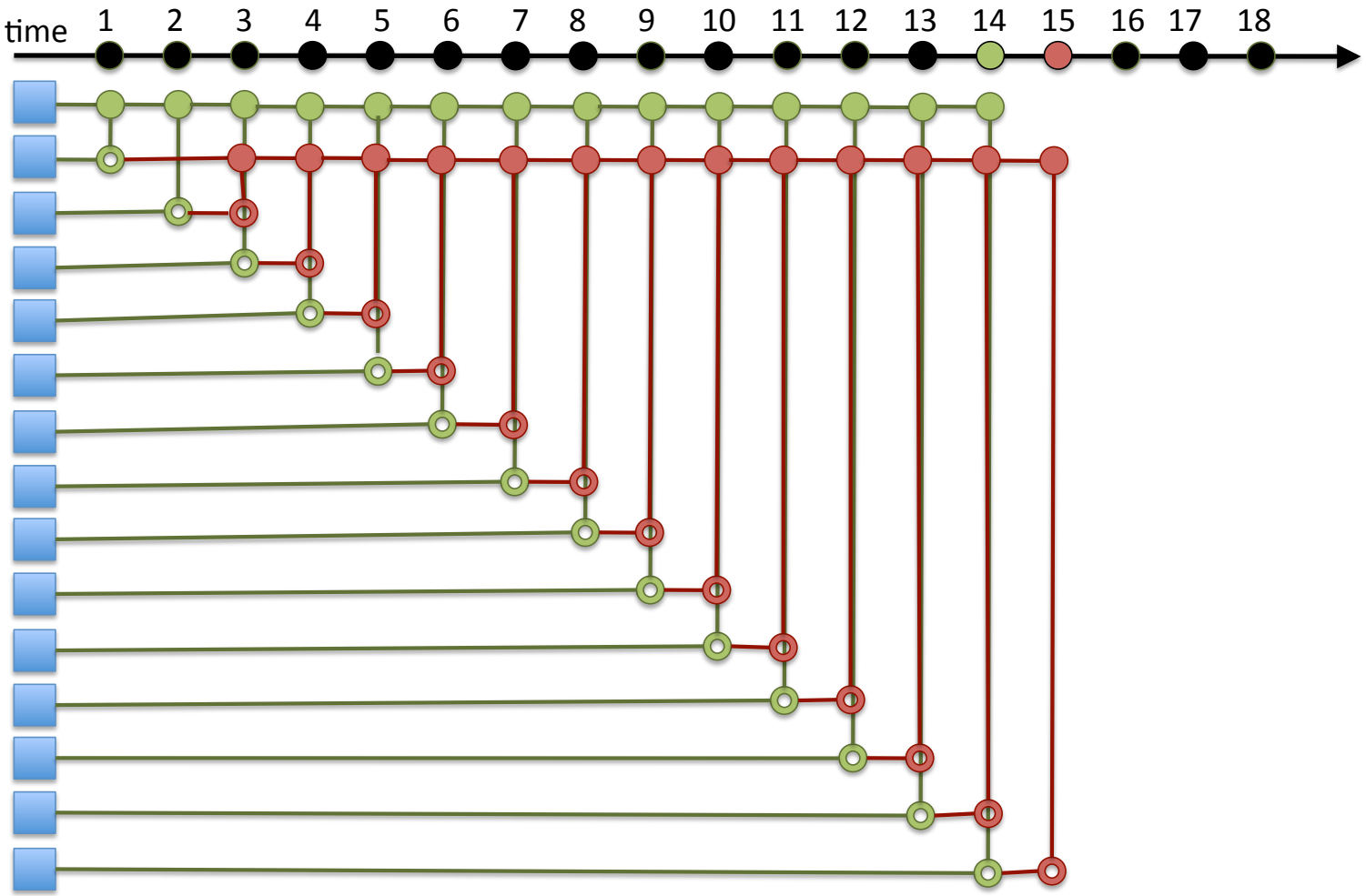
How well do you pipeline? BinaryTree



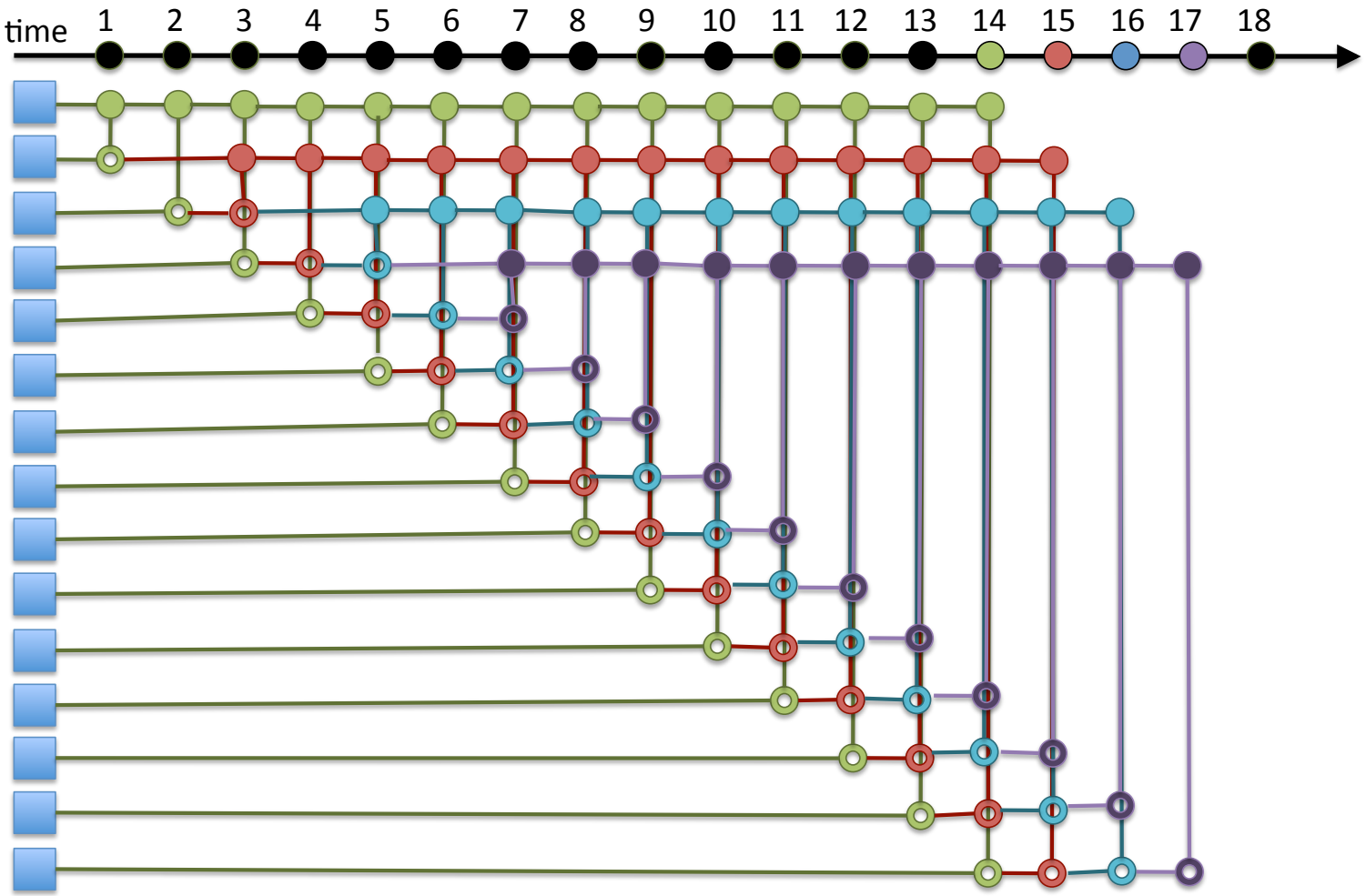
How well do you pipeline? FlatTree



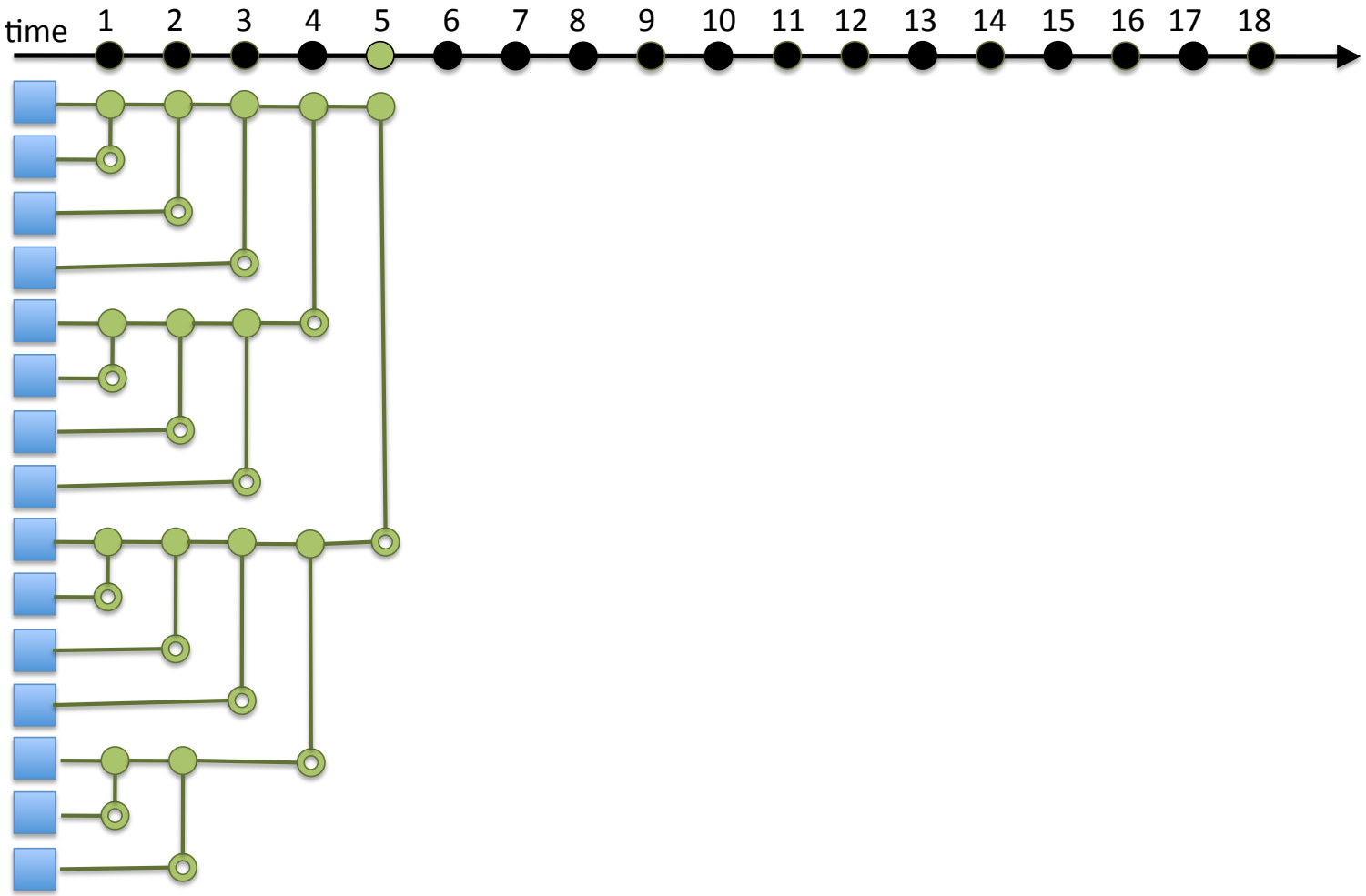
How well do you pipeline? FlatTree



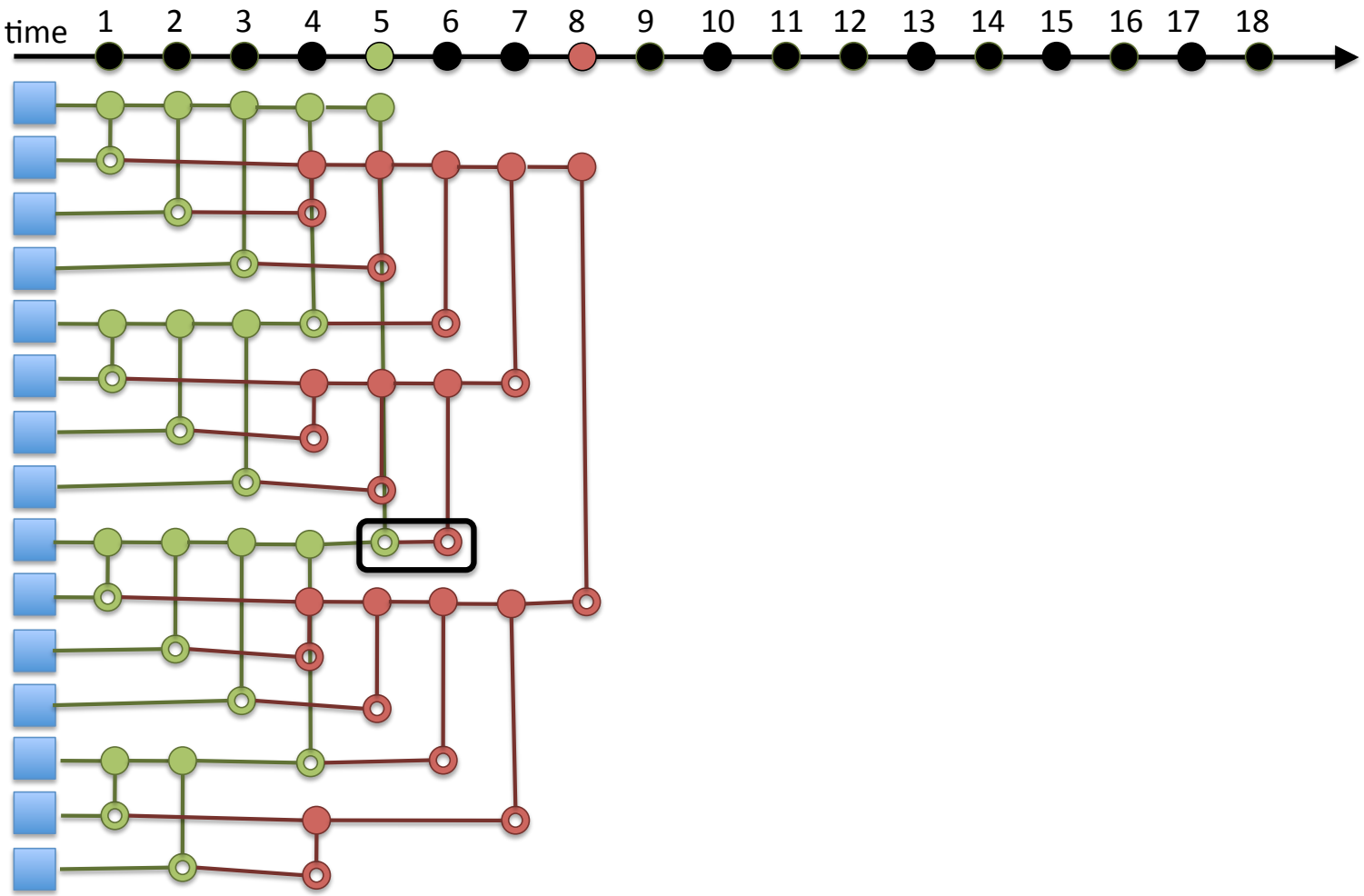
How well do you pipeline? FlatTree



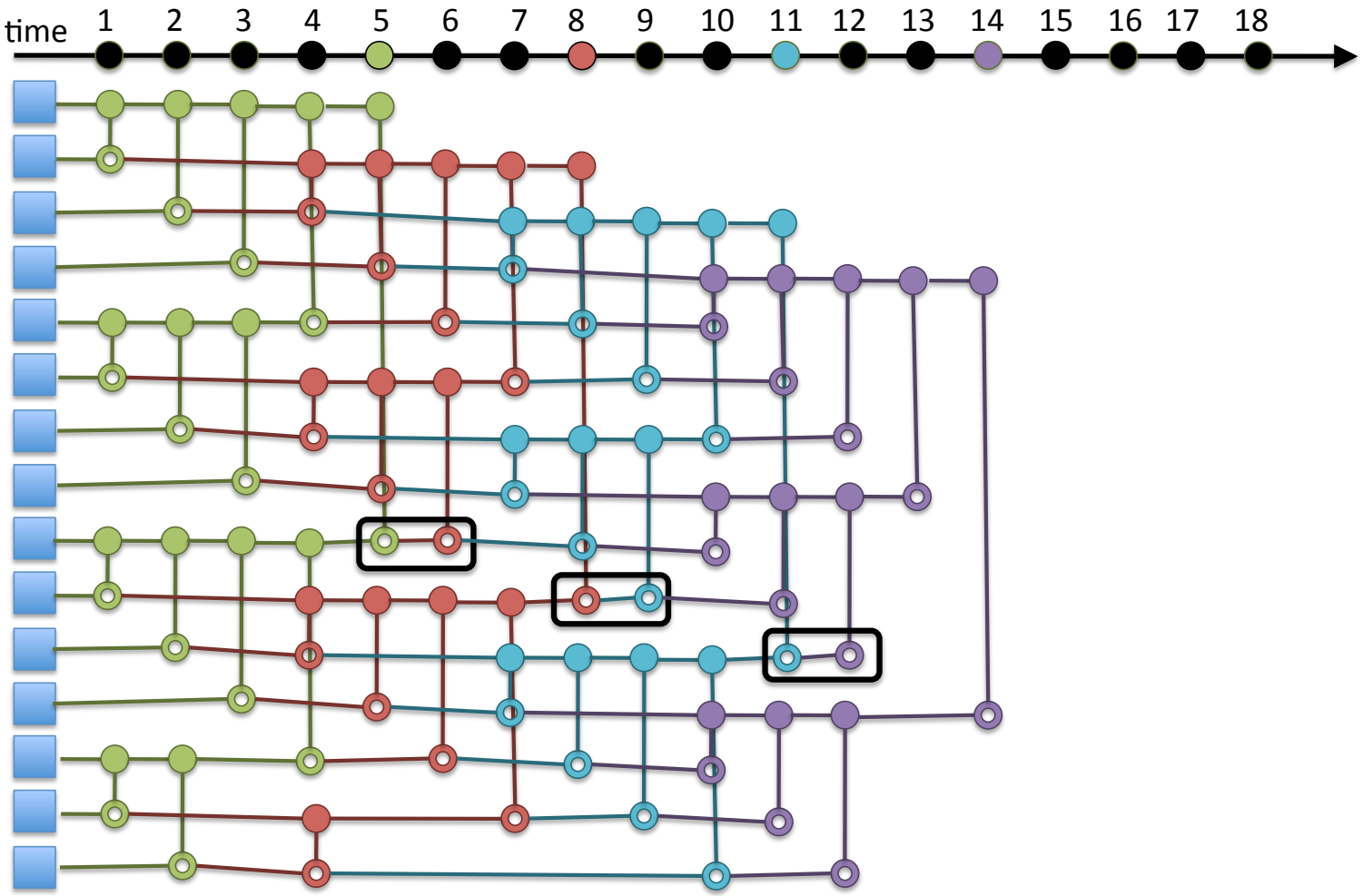
How well do you pipeline? PlasmaTree



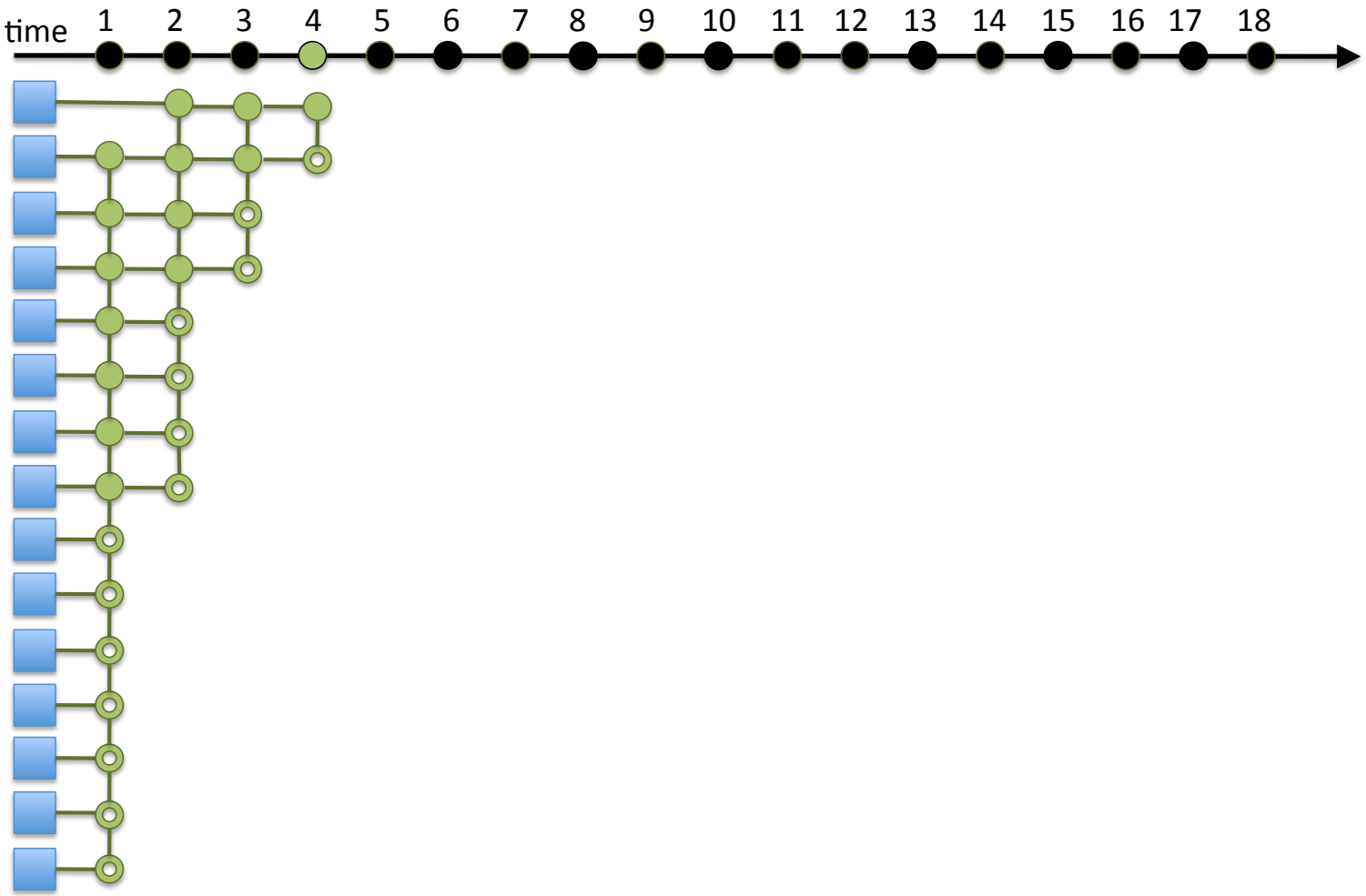
How well do you pipeline? PlasmaTree



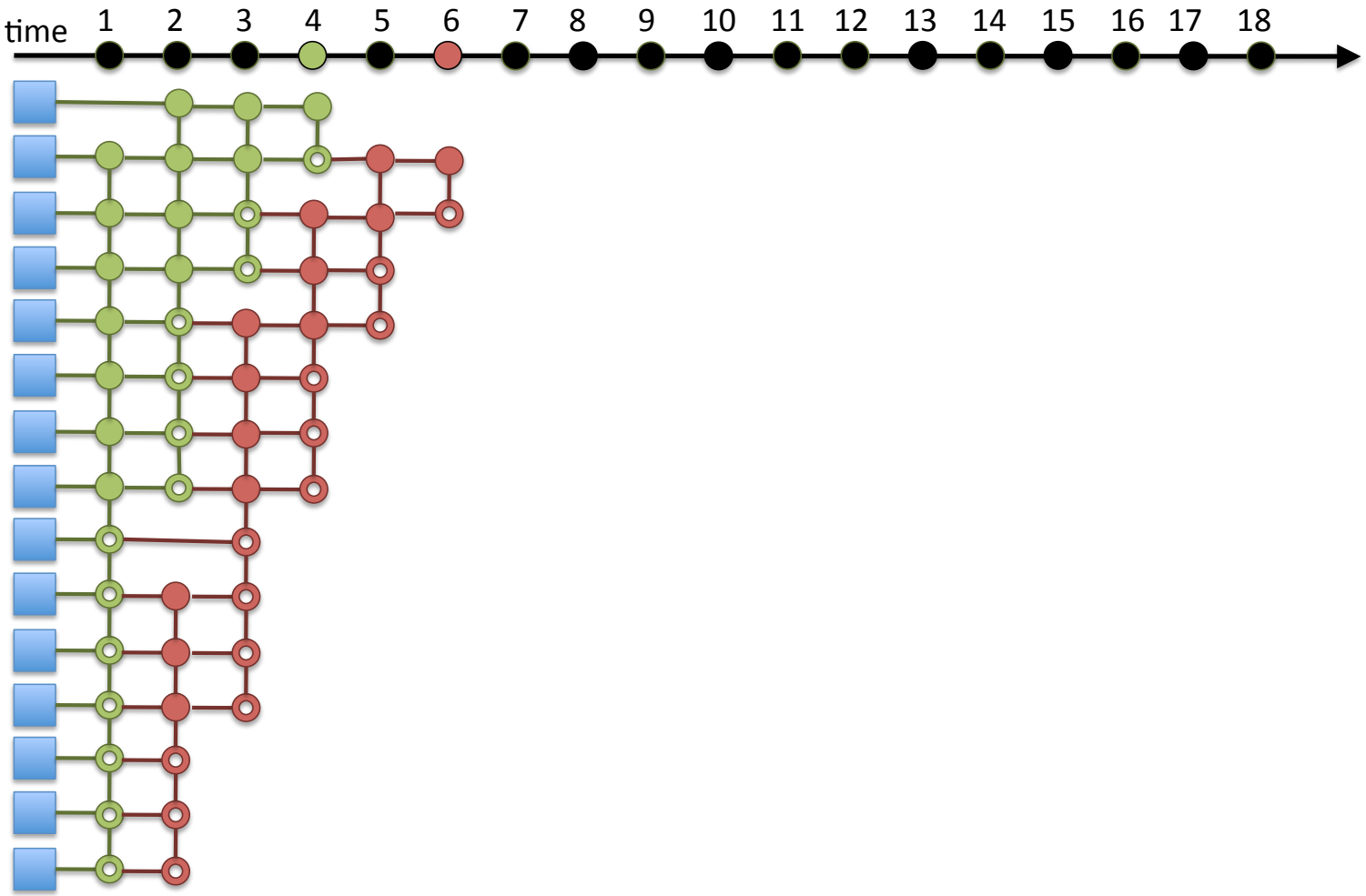
How well do you pipeline? PlasmaTree



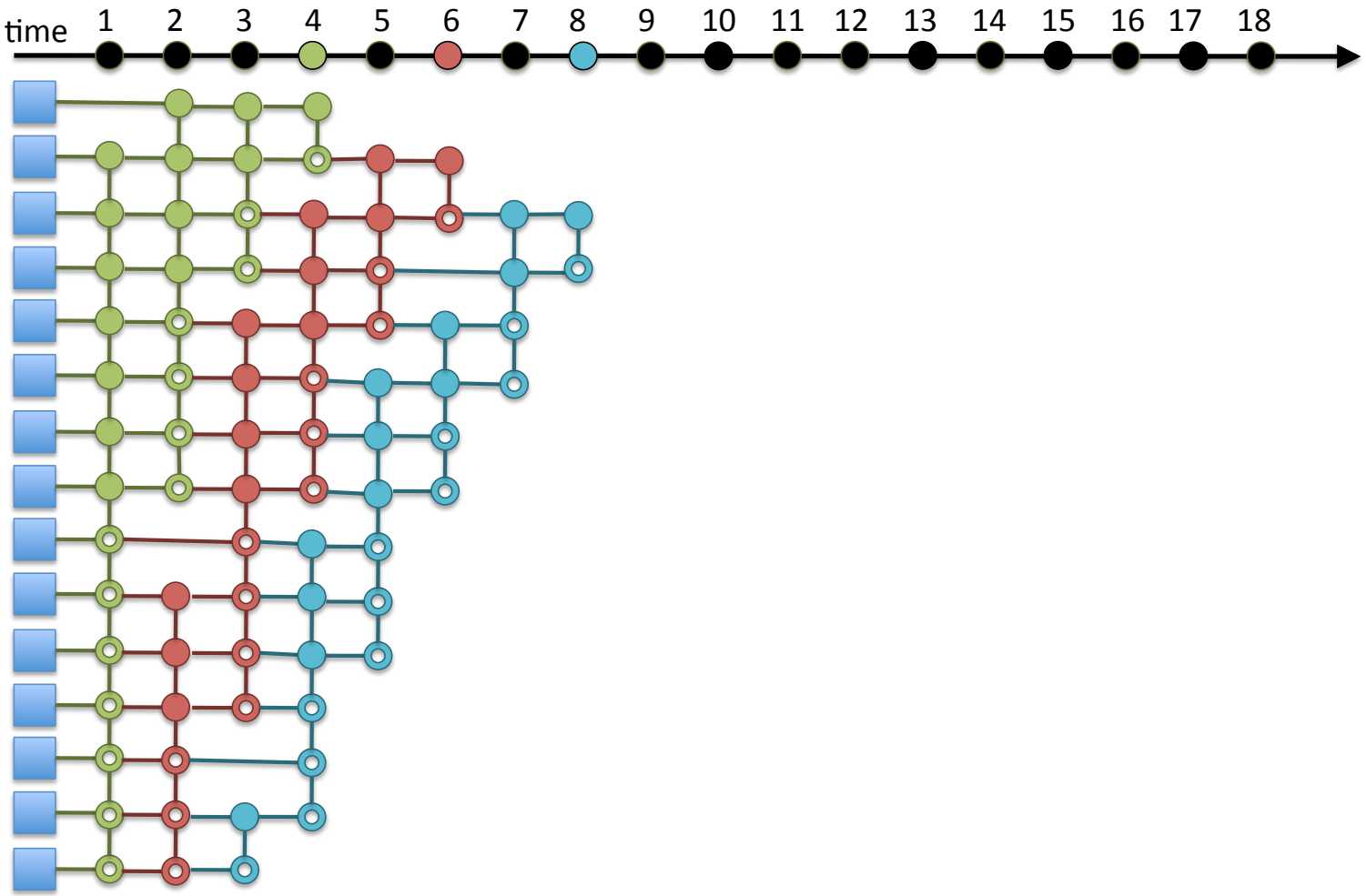
How well do you pipeline? Greedy



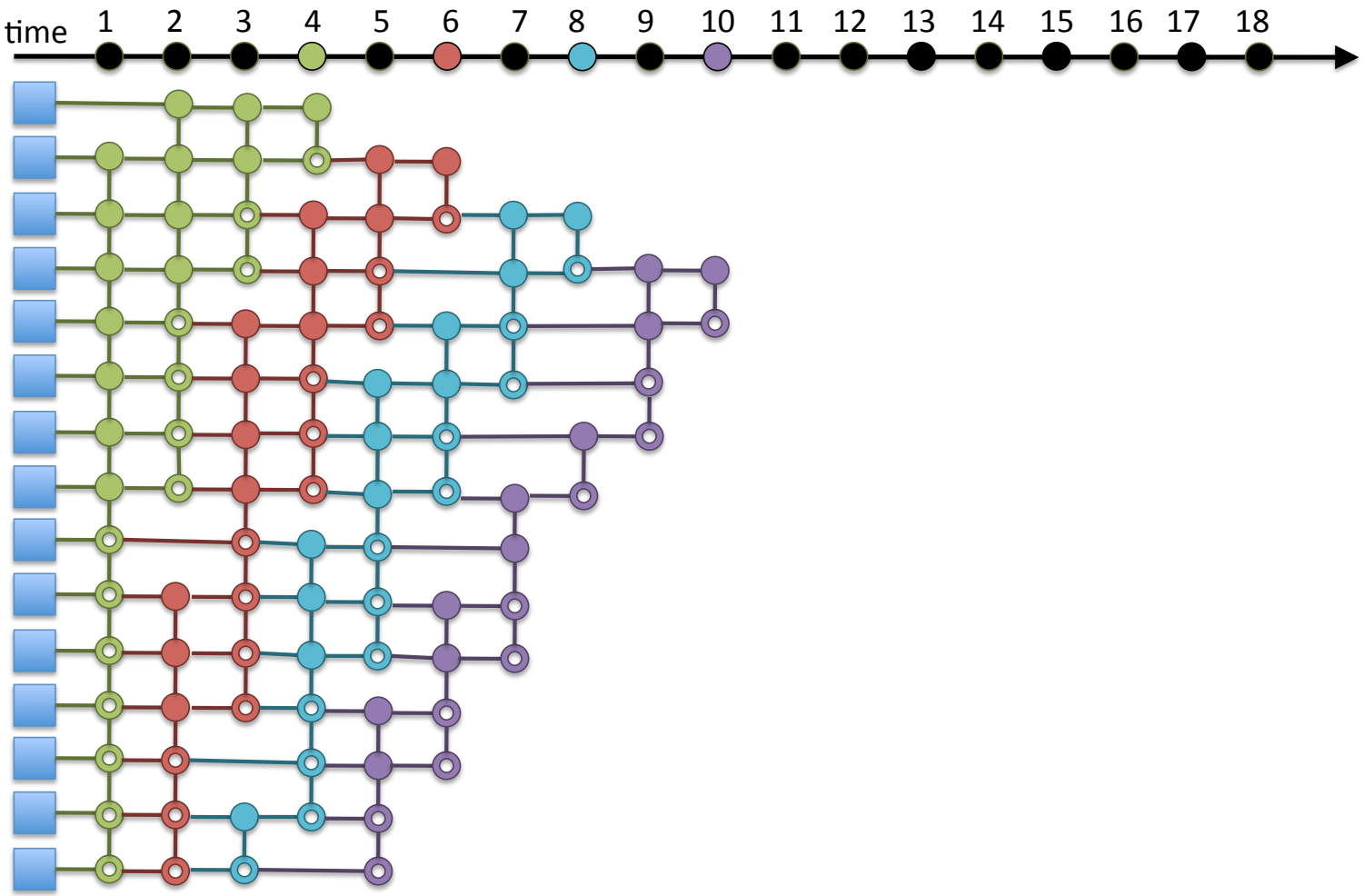
How well do you pipeline? Greedy



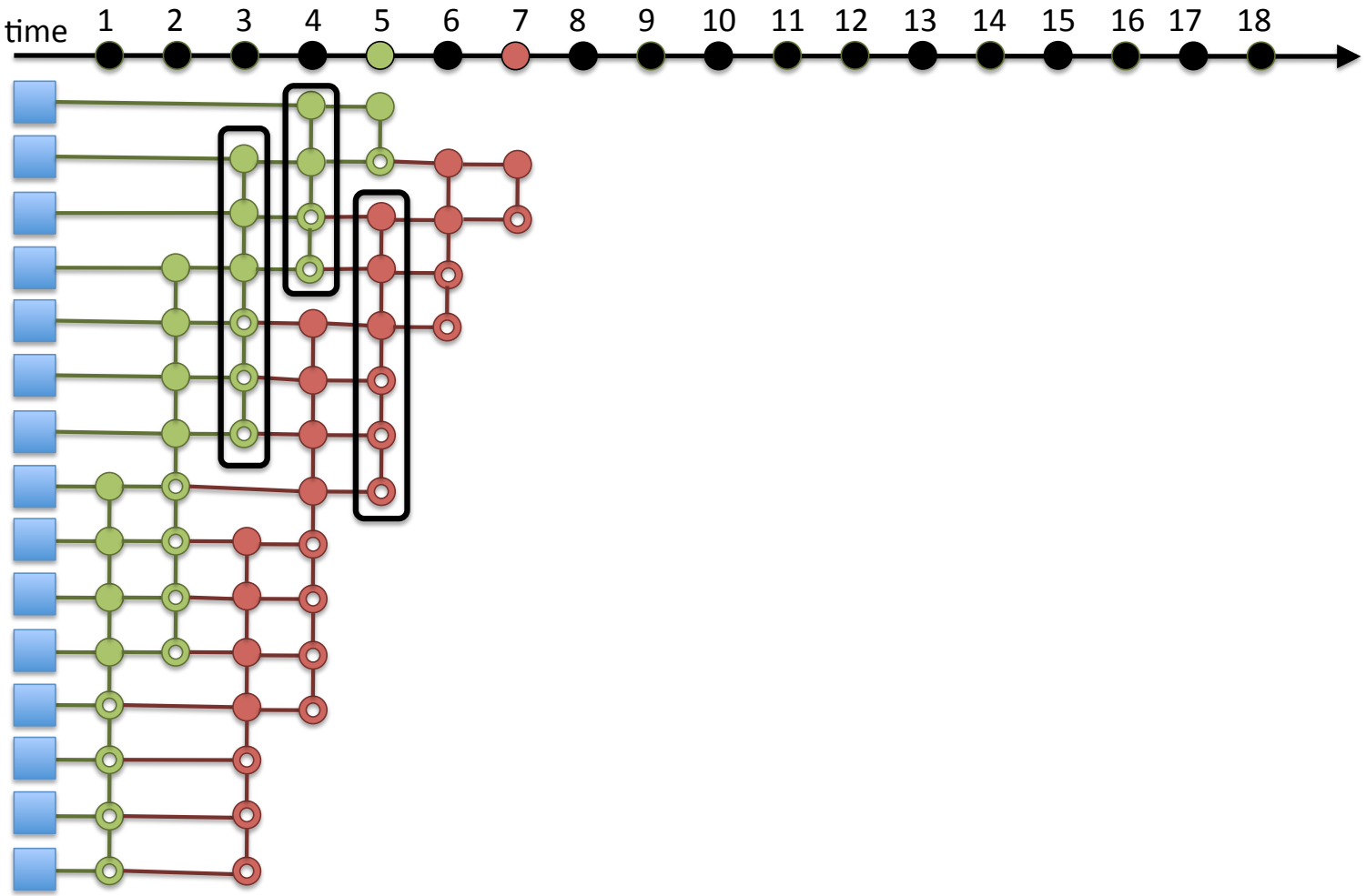
How well do you pipeline? Greedy



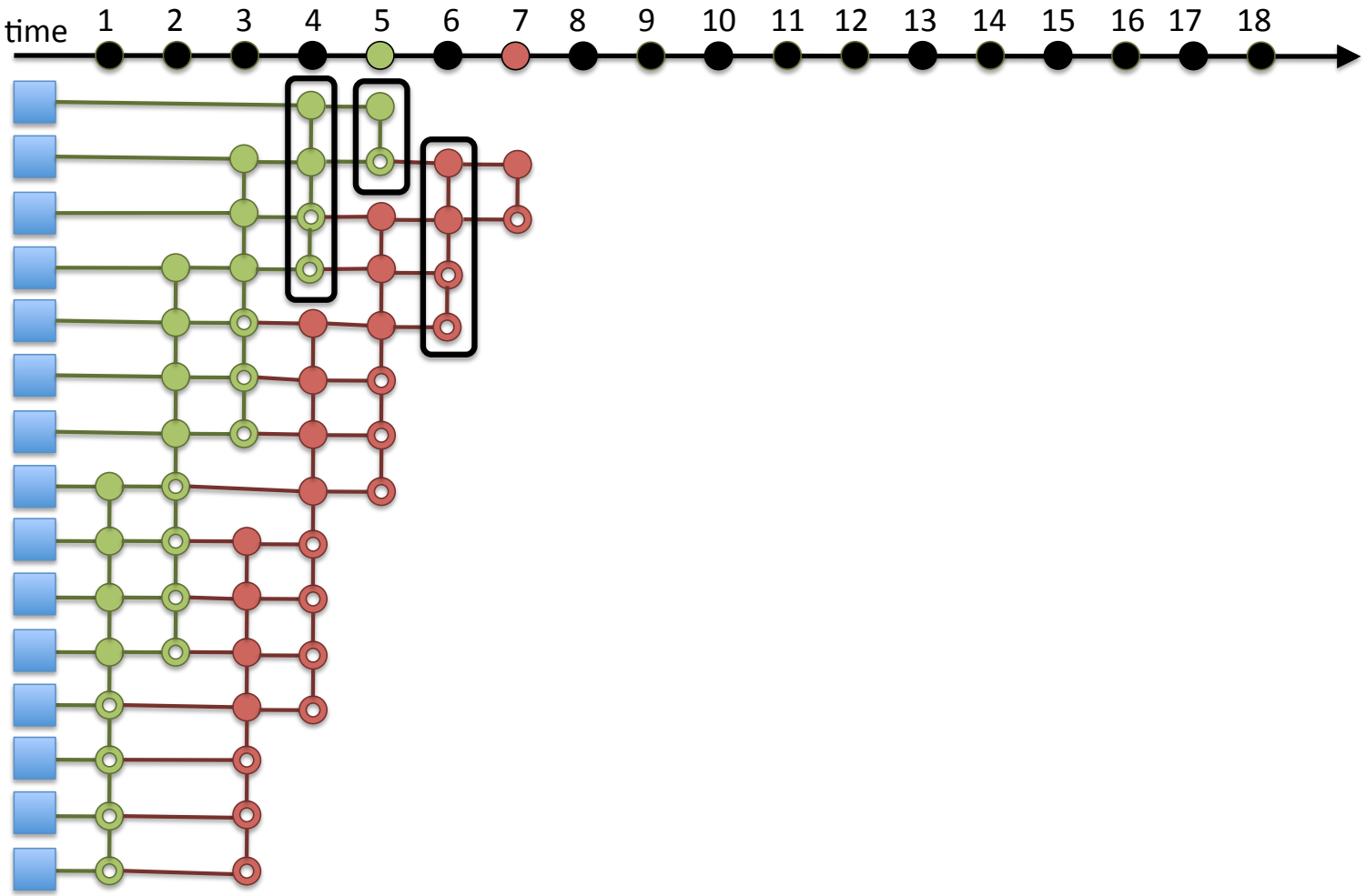
How well do you pipeline? Greedy



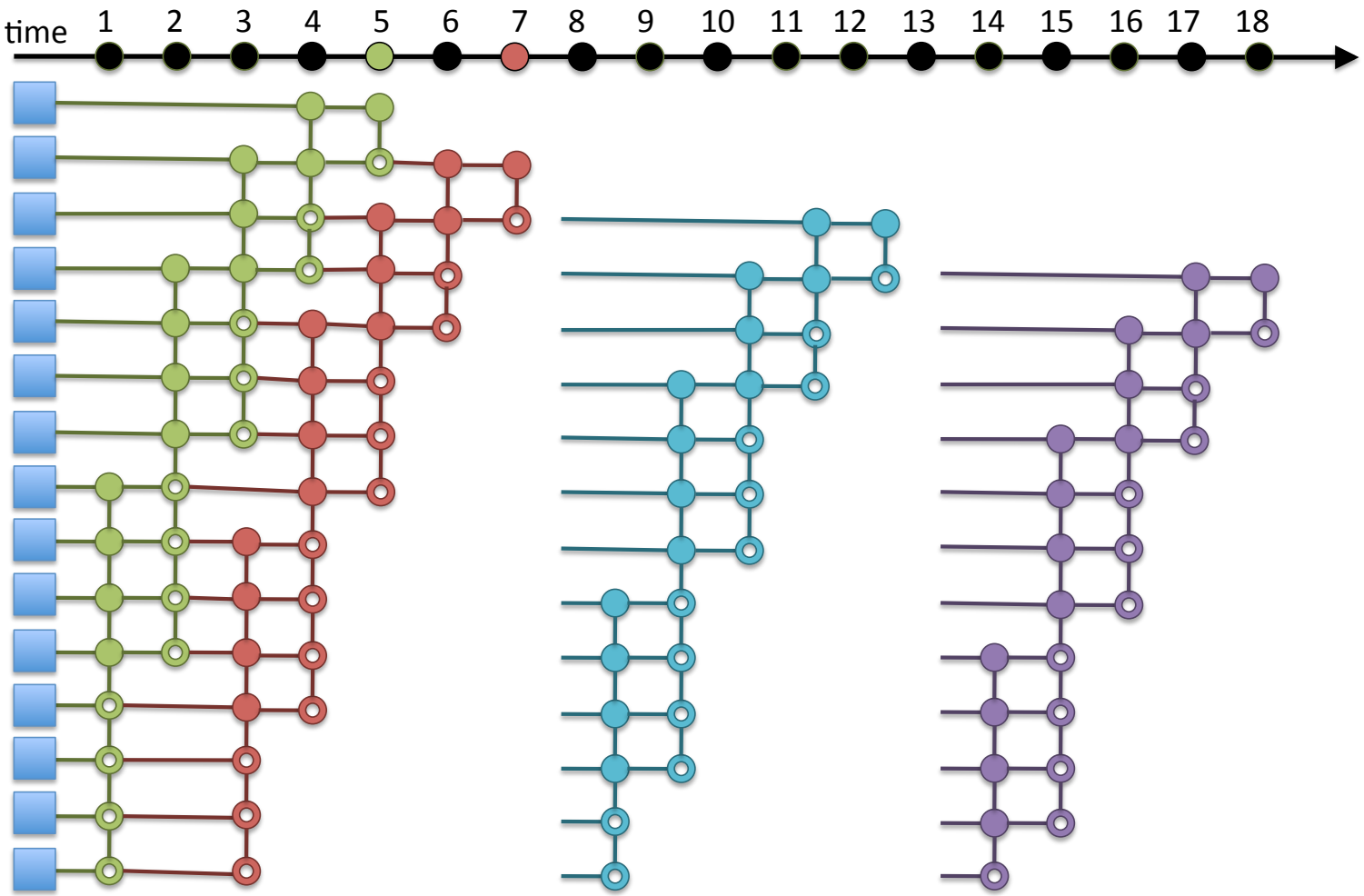
How well do you pipeline? Fibonacci



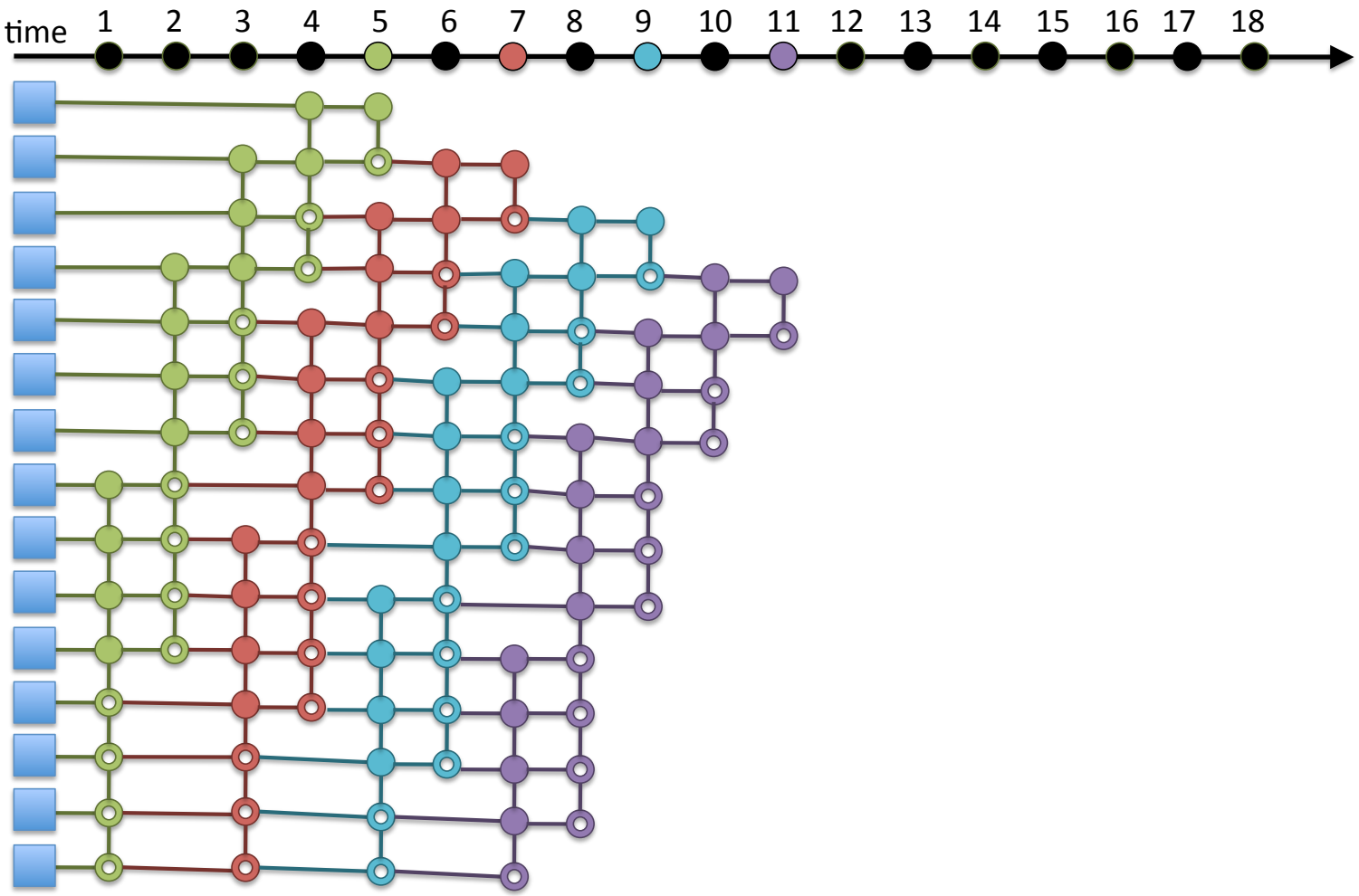
How well do you pipeline? Fibonacci



How well do you pipeline? Fibonacci



How well do you pipeline? Fibonacci



Outline

Tiled GREEDY algorithms

Algorithm 4: GREEDY algorithm via TT kernels.

for $j = 1$ *to* q **do**
  // $nz(j)$ is the number of tiles which have been eliminated in column j
  $nZ(j) = 0$
  // $nT(j)$ is the number of tiles which have been triangularized in column j
  $nT(j) = 0$

while *column q is not finished* **do**
  for $j = q$ *down to* 1 **do**
    if $j == 1$ **then**
      // Triangularize the first column if not yet done
      $nT_{new} = nT(j) + (p - nT(j))$
      if $p - nT(j) > 0$ **then**
        for $k = p$ *down to* 1 **do**
          $GEQRT(k, j)$
          for jj $= j + 1$ *to* q **do**
            $UNMQR(k, j, jj)$
    else
      // Triangularize tiles having a zero in the prev. column
      $nT_{new} = nZ(j-1)$
      for $k = nT(j)$ *to* $nT_{new} - 1$ **do**
        $GEQRT(p-k, j)$
        for jj $= j + 1$ *to* q **do**
          $UNMQR(p-k, j, jj)$
    // Eliminate tiles triangularized in the prev. step
    $nZ_{new} = nZ(j) + \lfloor \frac{nT(j) - nZ(j)}{2} \rfloor$
    for $kk = nZ(j)$ *to* $nZ_{new} - 1$ **do**
      $piv(p-kk) = p - kk - nZ_{new} + nZ(j)$
      $TTQRT(p-kk, piv(p-kk), j)$
      for jj $= j + 1$ *to* q **do**
        $TTMQR(p-kk, piv(p-kk), j, jj)$
    $nT(j) = nT_{new}$
    $nZ(j) = nZ_{new}$

Critical paths of tiled algorithms

- Coarse-grain algorithm $\Rightarrow$ tiled algorithm:
  - Keep same elimination list
  - Trigger each kernel as soon as possible
- Tiled version of SAMEH-KUCK is FLATTREE in PLASMA

(a) SAMEH-KUCK						(b) FIBONACCI						(c) GREEDY						(d) BINARYTREE						(e) PLASMATREE ($BS = 5$)					
$\star$						$\star$						$\star$						$\star$						$\star$					
6	$\star$					14	$\star$					12	$\star$					6	$\star$					6	$\star$				
8	28	$\star$				12	48	$\star$				10	42	$\star$				8	28	$\star$				8	28	$\star$			
10	34	50	$\star$			12	46	70	$\star$			10	40	64	$\star$			6	36	56	$\star$			10	34	50	$\star$		
12	40	56	72	$\star$		10	42	68	92	$\star$		8	36	62	86	$\star$		10	34	70	90	$\star$		12	40	56	72	$\star$	
14	46	62	78	94	$\star$	10	40	64	90	114	$\star$	8	34	56	84	106	$\star$	6	44	68	104	124	$\star$	14	46	62	78	94	$\star$
16	52	68	84	100	116	10	40	62	86	112	136	8	34	56	78	102	128	8	28	78	102	138	158	6	54	74	90	106	122
18	58	74	90	106	122	8	36	62	84	108	134	8	30	52	78	100	122	6	42	62	112	136	172	8	28	82	102	118	134
20	64	80	96	112	128	8	34	58	84	106	130	6	28	50	72	100	118	12	40	76	96	146	170	10	34	50	110	130	146
22	70	86	102	118	134	8	34	56	80	106	128	6	28	50	72	94	116	6	46	74	110	130	180	12	40	56	72	138	158
24	76	92	108	124	140	8	34	56	78	102	128	6	28	50	68	94	116	8	28	80	108	144	164	16	52	68	84	100	166
26	82	98	114	130	146	6	28	56	78	100	122	6	28	44	66	88	110	6	36	56	114	142	178	6	56	80	96	112	128
28	88	104	120	136	152	6	28	50	78	100	122	6	22	44	66	88	110	10	34	64	84	148	176	8	28	84	108	124	140
30	94	110	126	142	158	6	28	44	72	100	122	6	22	44	60	82	104	6	38	62	92	112	182	10	34	50	112	136	152
32	100	116	132	148	164	6	22	44	60	94	116	6	22	38	60	76	98	8	28	66	90	114	134	12	40	56	72	140	164

Critical paths of tiled algorithms

(a) SAMEH-KUCK

⋆					
6	⋆				
8	28	⋆			
10	34	50	⋆		
12	40	56	72	⋆	
14	46	62	78	94	⋆
16	52	68	84	100	116
18	58	74	90	106	122
20	64	80	96	112	128
22	70	86	102	118	134
24	76	92	108	124	140
26	82	98	114	130	146
28	88	104	120	136	152
30	94	110	126	142	158
32	100	116	132	148	164

Critical paths of tiled algorithms

(b) FIBONACCI

$\star$					
14	$\star$				
12	48	$\star$			
12	46	70	$\star$		
10	42	68	92	$\star$	
10	40	64	90	114	$\star$
10	40	62	86	112	136
8	36	62	84	108	134
8	34	58	84	106	130
8	34	56	80	106	128
8	34	56	78	102	128
6	28	56	78	100	122
6	28	50	78	100	122
6	28	44	72	100	122
6	22	44	60	94	116

Critical paths of tiled algorithms

(c) GREEDY					
$\star$					
12	$\star$				
10	42	$\star$			
10	40	64	$\star$		
8	36	62	86	$\star$	
8	34	56	84	106	$\star$
8	34	56	78	102	128
8	30	52	78	100	122
6	28	50	72	100	118
6	28	50	72	94	116
6	28	50	68	94	116
6	28	44	66	88	110
6	22	44	66	88	110
6	22	44	60	82	104
6	22	38	60	76	98

Critical paths of tiled algorithms

(d) BinaryTree					
$\star$					
6	$\star$				
8	28	$\star$			
6	36	56	$\star$		
10	34	70	90	$\star$	
6	44	68	104	124	$\star$
8	28	78	102	138	158
6	42	62	112	136	172
12	40	76	96	146	170
6	46	74	110	130	180
8	28	80	108	144	164
6	36	56	114	142	178
10	34	64	84	148	176
6	38	62	92	112	182
8	28	66	90	114	134

Critical paths of tiled algorithms

(e) PlasmaTree ($BS = 5$)					
$\star$					
6	$\star$				
8	28	$\star$			
10	34	50	$\star$		
12	40	56	72	$\star$	
14	46	62	78	94	$\star$
6	54	74	90	106	122
8	28	82	102	118	134
10	34	50	110	130	146
12	40	56	72	138	158
16	52	68	84	100	166
6	56	80	96	112	128
8	28	84	108	124	140
10	34	50	112	136	152
12	40	56	72	140	164

Neither GREEDY nor ASAP optimal

- ASAP eliminates a tile as soon as there are two rows ready

(a) GREEDY		(b) ASAP	
$\star$		$\star$	
12	$\star$	12	$\star$
10	42	10	40
10	40	10	36
8	36	8	34
8	34	8	32
8	34	8	30
8	30	8	28
6	28	6	28
6	28	6	26
6	28	6	24
6	28	6	24
6	22	6	22
6	22	6	22
6	22	6	22

Time steps of GREEDY and ASAP for a 15×2 matrix

Neither GREEDY nor ASAP optimal

- ASAP eliminates a tile as soon as there are two rows ready

(a) GREEDY			(b) ASAP		
$\star$			$\star$		
12	$\star$		12	$\star$	
10	42	$\star$	10	40	$\star$
10	40	64	10	36	86
8	36	62	8	34	80
8	34	56	8	32	74
8	34	56	8	30	68
8	30	52	8	28	62
6	28	50	6	28	56
6	28	50	6	26	50
6	28	50	6	24	46
6	28	44	6	24	44
6	22	44	6	22	44
6	22	44	6	22	40
6	22	38	6	22	38

Time steps of GREEDY and ASAP for a 15×3 matrix

Neither GREEDY nor ASAP optimal

- ASAP eliminates a tile as soon as there are two rows ready

(a) GREEDY			(b) ASAP		
$\star$			$\star$		
12	$\star$		12	$\star$	
10	42	$\star$	10	40	$\star$
10	40	64	10	36	86
8	36	62	8	34	80
8	34	56	8	32	74
8	34	56	8	30	68
8	30	52	8	28	62
6	28	50	6	28	56
6	28	50	6	26	50
6	28	50	6	24	46
6	28	44	6	24	44
6	22	44	6	22	44
6	22	44	6	22	40
6	22	38	6	22	38

Time steps of GREEDY and ASAP for a 15×3 matrix

		q			
p	Algorithm	16	32	64	128
16	GREEDY	310			
	ASAP	310			
32	GREEDY	360	650		
	ASAP	402	656		
64	GREEDY	374	726	1342	
	ASAP	588	844	1354	
128	GREEDY	396	748	1452	2732
	ASAP	966	1222	1748	2756

GREEDY generally outperforms ASAP.

Critical paths of tiled algorithms

Theorem

Consider a tiled matrix of size $p \times q$, where $p \geq q$:

① The critical path length of FlatTree is

$$\begin{cases} 2p + 2 & \text{if } p \geq q = 1 \\ 6p + 16q - 22 & \text{if } p > q > 1 \\ 22q - 24 & \text{if } p = q > 1 \end{cases}$$

Critical paths of tiled algorithms

Theorem

Consider a tiled matrix of size $p \times q$, where $p \geq q$:

① The critical path length of FlatTree is

$$\begin{cases} 2p+2 & \text{if } p \geq q = 1 \\ 6p+16q-22 & \text{if } p > q > 1 \\ 22q-24 & \text{if } p = q > 1 \end{cases}$$

Proof

By induction on k

Tile (i,k) zeroed out at time $6i+16k-22$

Critical paths of tiled algorithms

Theorem

Consider a tiled matrix of size $p \times q$, where $p \geq q$:

② The critical path length of FIBONACCI is $\leq 22q + 6\lceil\sqrt{2p}\rceil$

Critical paths of tiled algorithms

Theorem

Consider a tiled matrix of size $p \times q$, where $p \geq q$:

② The critical path length of FIBONACCI is $\leq 22q + 6\lceil\sqrt{2p}\rceil$

Proof

- $coarse(i, k)$: time-step to zero-out (i, k) in coarse-grain model
- Derive "slowed down" version of tiled algorithm
- Completion of (i, k) at time-step

$$6coarse(2, 1) + 22(k - 1) - 6(coarse(k, k) - coarse(i, k))$$

- Proof by induction on k that dependences are enforced
- Sanity check by program verification

Critical paths of tiled algorithms

Theorem

Consider a tiled matrix of size $p \times q$, where $p \geq q$:

③ The critical path length of GREEDY is $\leq 22q + 6\lceil \log_2 p \rceil$

Critical paths of tiled algorithms

Theorem

Consider a tiled matrix of size $p \times q$, where $p \geq q$:

③ The critical path length of GREEDY is $\leq 22q + 6\lceil \log_2 p \rceil$

Proof

Same proof, modify *coarse*(i, k) for GREEDY

Critical paths of tiled algorithms

Theorem

Consider a tiled matrix of size $p \times q$, where $p \geq q$:

④ The optimal critical path length is $\geq 22q - 30$

Critical paths of tiled algorithms

Theorem

Consider a tiled matrix of size $p \times q$, where $p \geq q$:

④ The optimal critical path length is $\geq 22q - 30$

Proof

- Tricky proof ☺
- Consider $q \times q$ matrix with three sub-diagonals
- Try all possible row pairings for, say, 5 first columns
- Identify repeating pattern for fastest progression
- Sanity check by program verification

Critical paths of tiled algorithms

Theorem

Consider a tiled matrix of size $p \times q$, where $p \geq q$:

⑤ FIBONACCI is asymptotically optimal if $p = q^2 f(q)$
⑥ GREEDY is asymptotically optimal if $\log_2 p = qf(q)$
for any function f s.t. $\lim_{+\infty} f = 0$

Critical paths of tiled algorithms

Theorem

Consider a tiled matrix of size $p \times q$, where $p \geq q$:

⑤ FIBONACCI is asymptotically optimal if $p = q^2 f(q)$
⑥ GREEDY is asymptotically optimal if $\log_2 p = qf(q)$
for any function f s.t. $\lim_{+\infty} f = 0$

Proof

Direct consequence of ② ③ ④

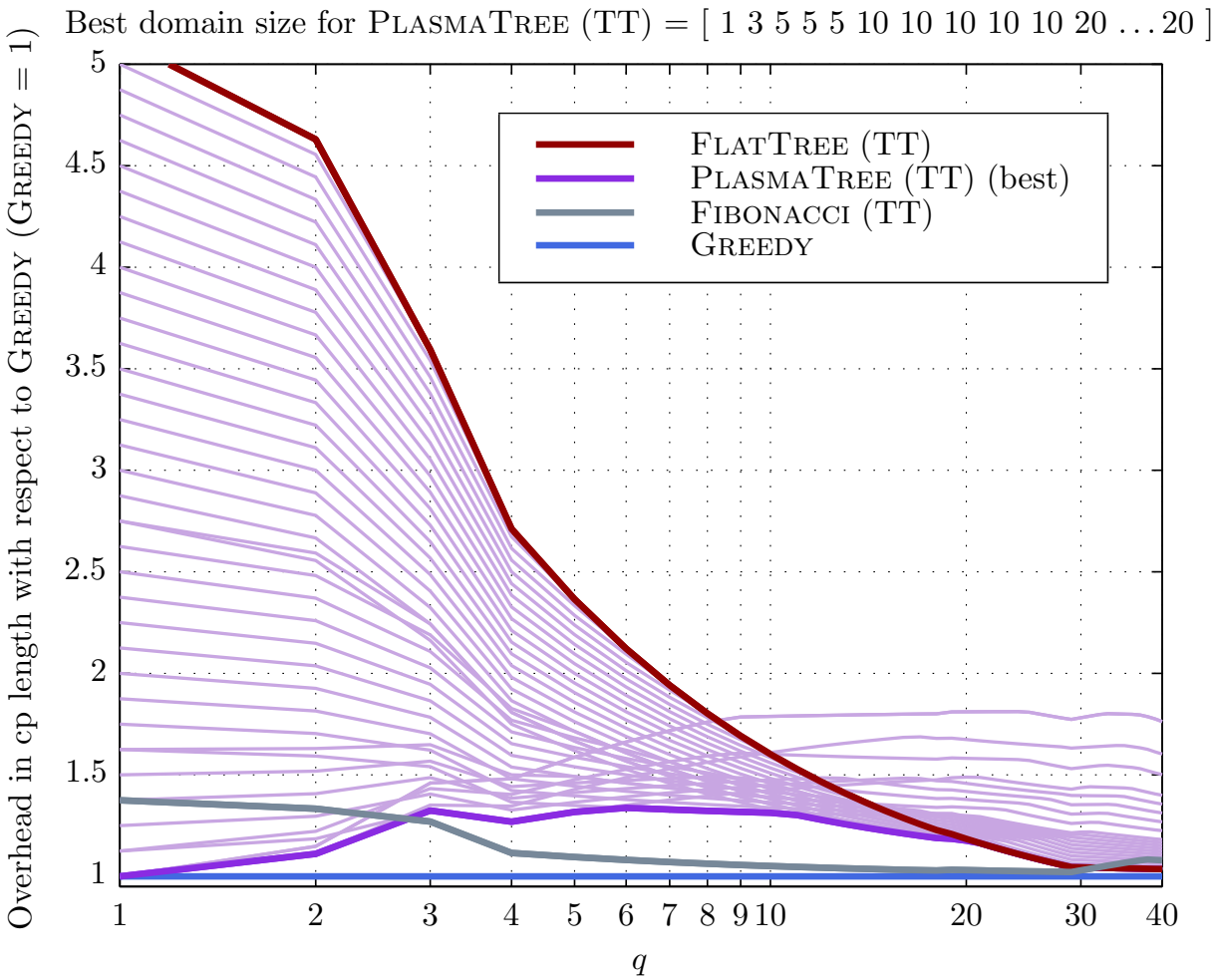
Outline

Platform

- 48-core machine composed of eight hexa-core AMD Opteron 8439 SE (codename Istanbul)
- Processors running at 2.8 GHz
- Each core: theoretical peak of 11.2 Gflop/s
- Peak of 537.6 Gflop/s for whole machine
- Experimental code written using QUARK scheduler

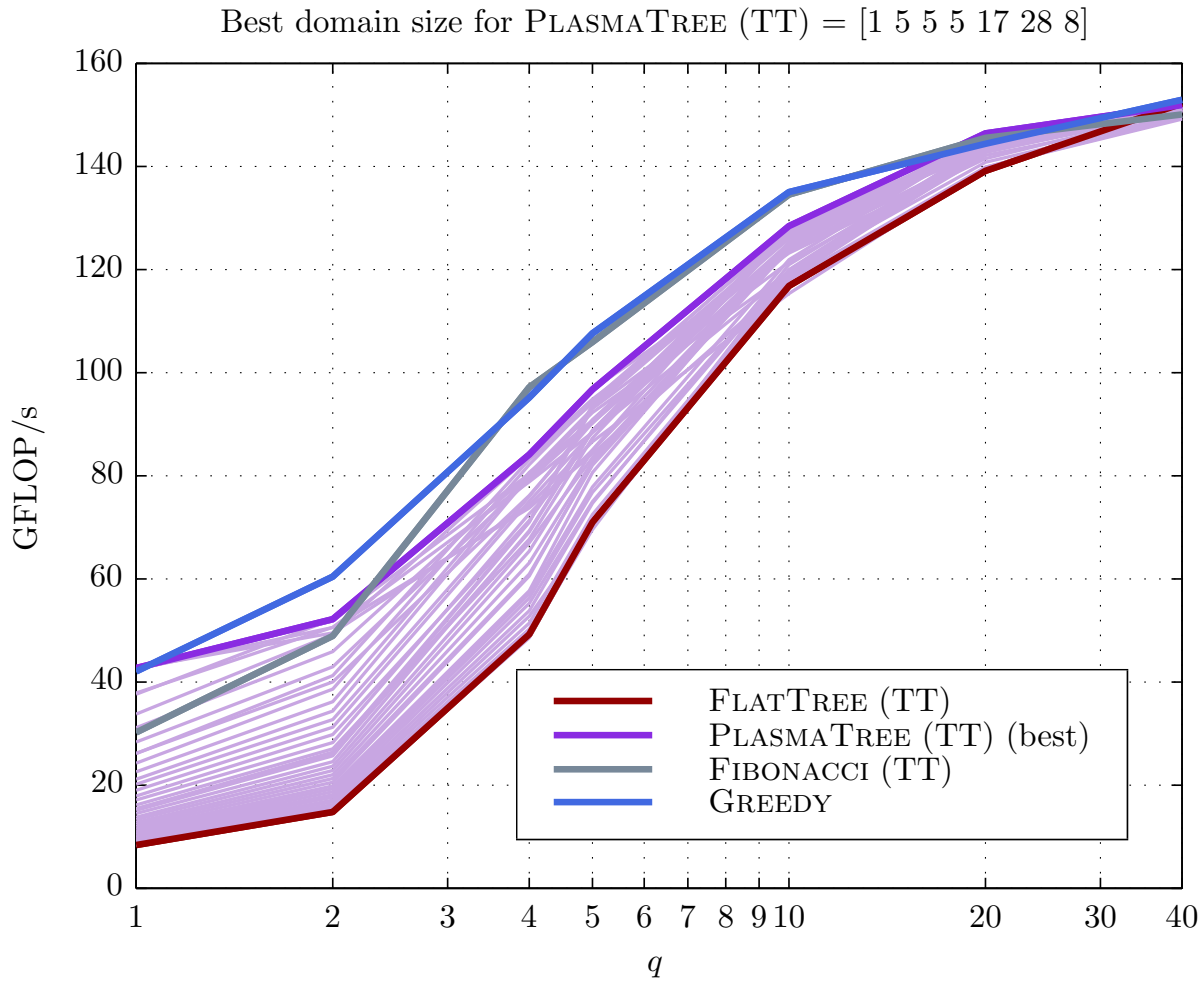
Overhead in critical path lengths (TT kernels only)

$$P = 40,\ Q \in [1..40]$$



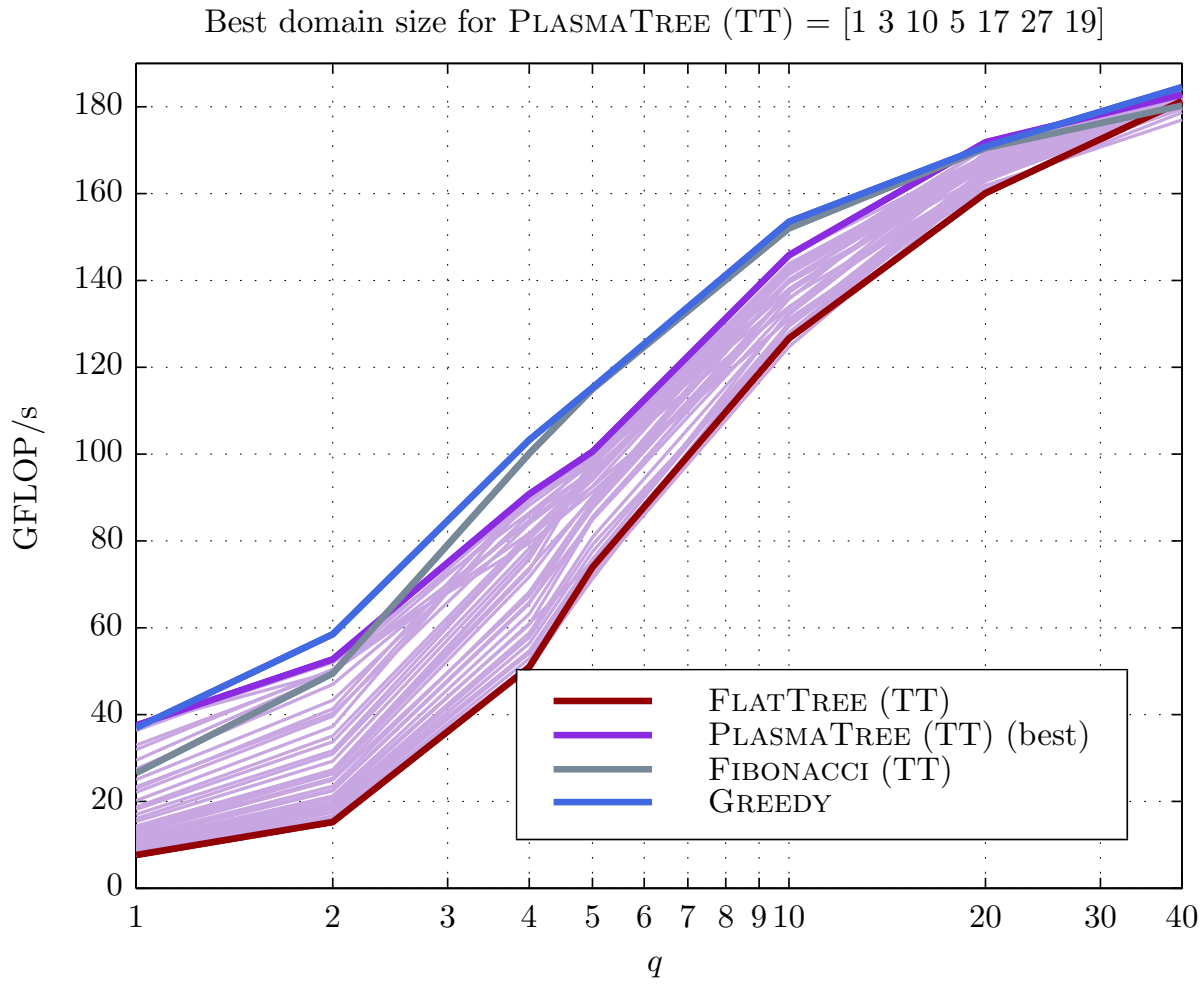
Running time (double complex precision) (TT kernels only)

$$P = 40,\ Q \in [1..40]$$



Running time (double precision) (TT kernels only)

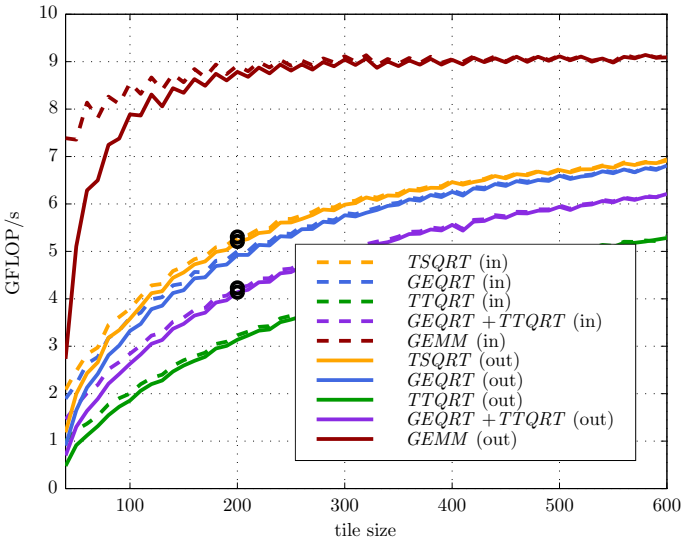
$$P = 40,\ Q \in [1..40]$$



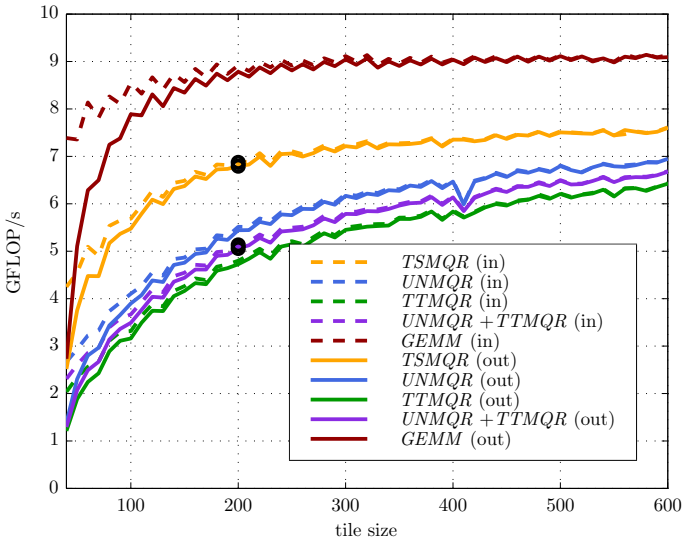
TT kernels vs. TS kernels performance

For double complex precision :

Factorization kernels



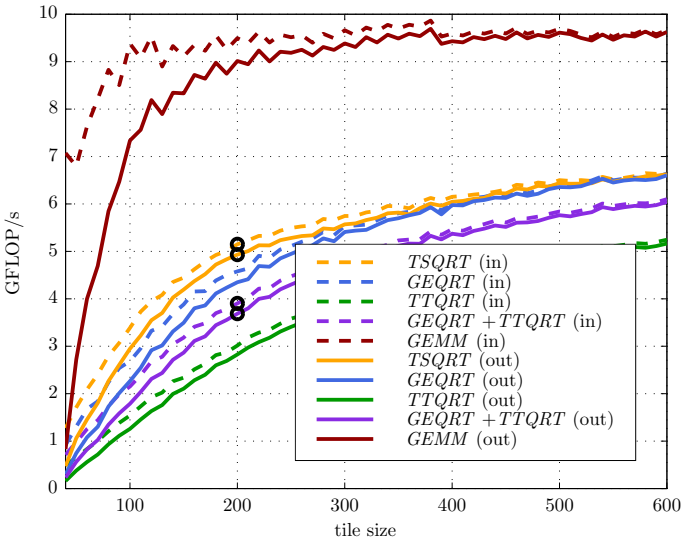
Update kernels



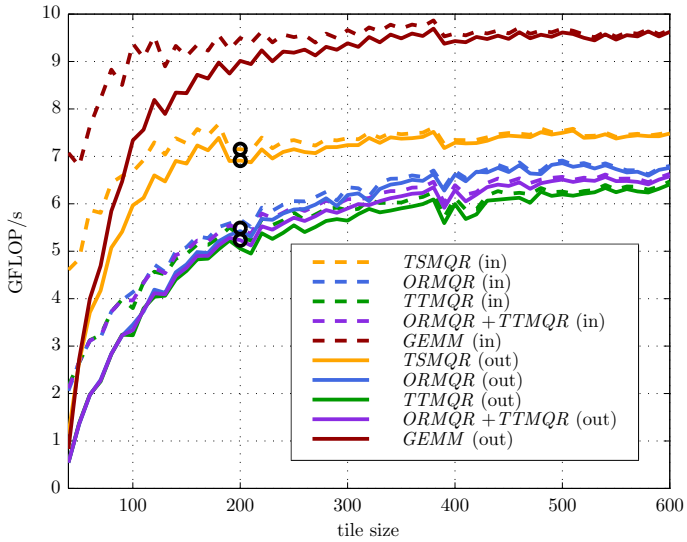
TT kernels vs. TS kernels performance

For double precision :

Factorization kernels

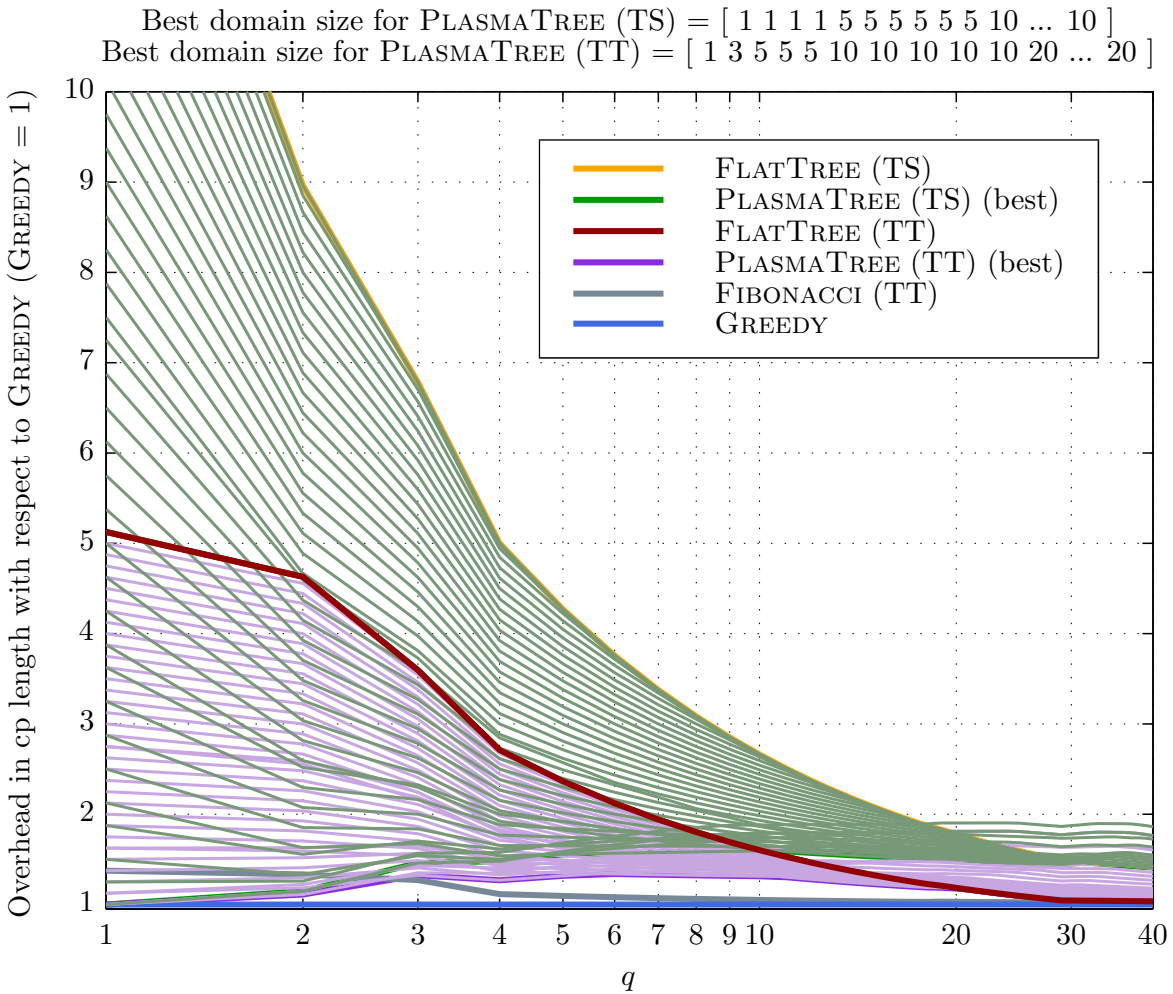


Update kernels



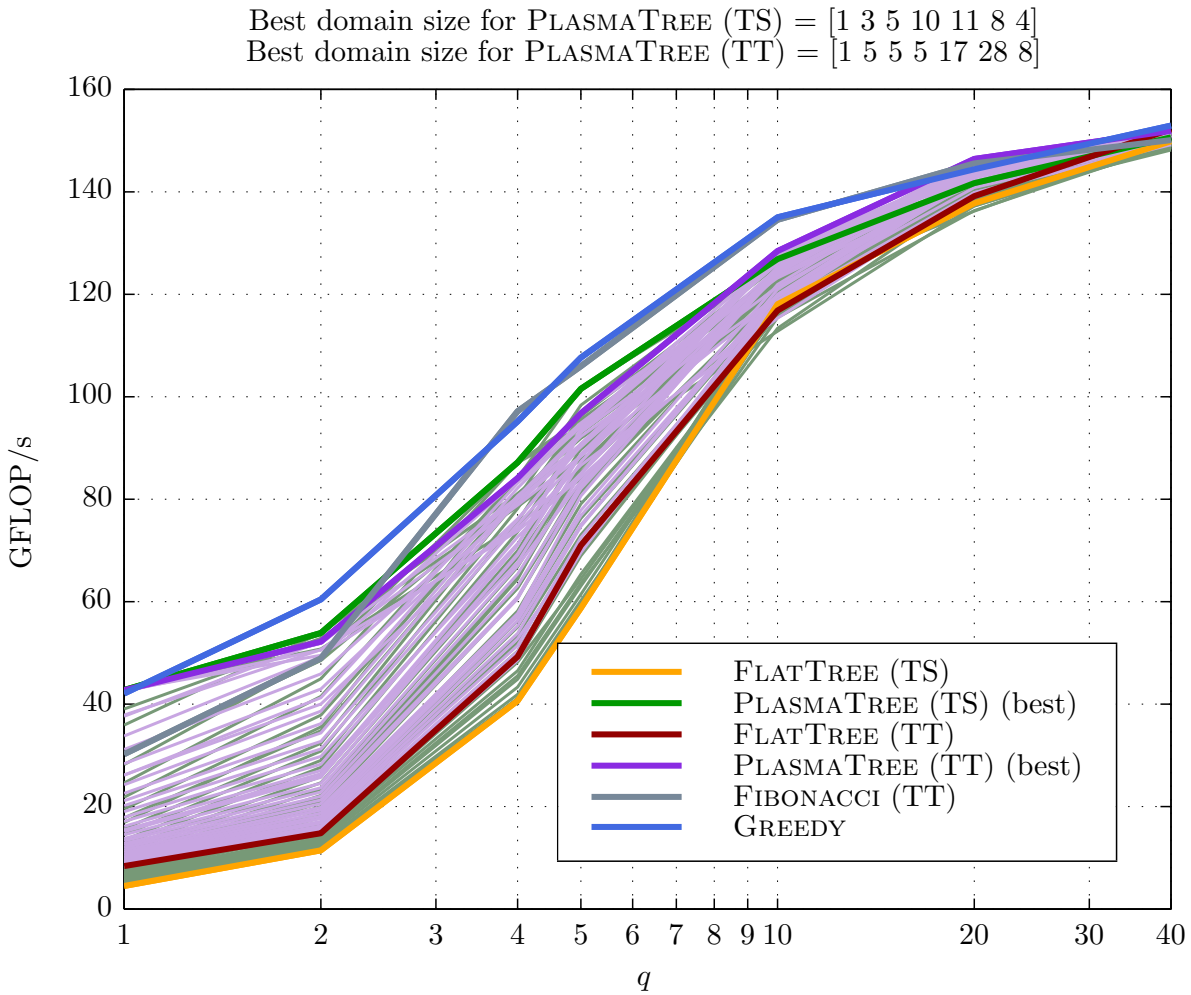
Overhead in critical path lengths (All kernels)

$$P = 40,\ Q \in [1..40]$$



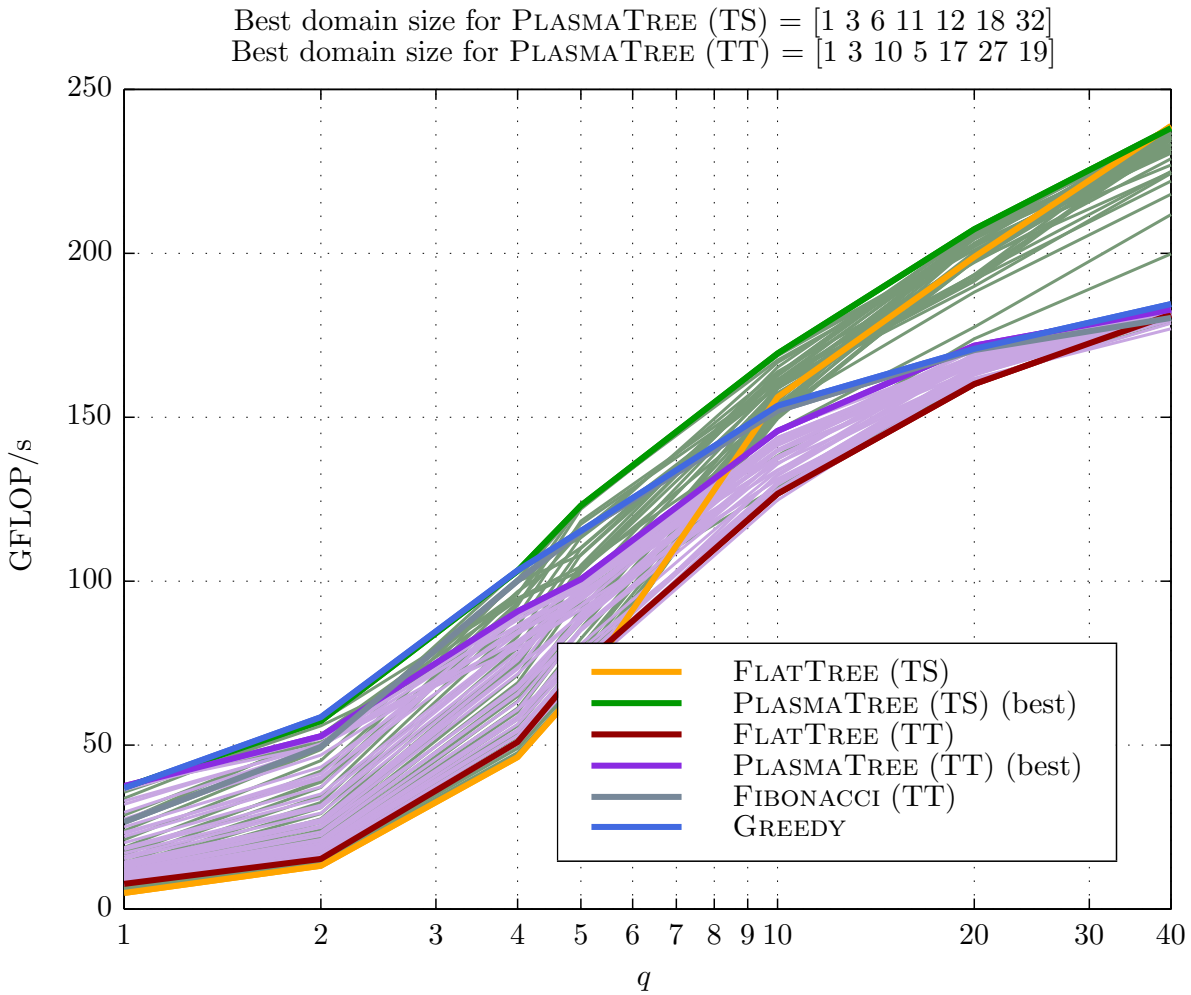
Running time (double complex precision) (All kernels)

$$P = 40,\ Q \in [1..40]$$



Running time (double precision) (All kernels)

$$P = 40,\ Q \in [1..40]$$



Conclusion

- Design of new tiled algorithms
- Exact or asymptotic formulas for their critical paths
- SimGrid simulator helpful!
- Experiments prove superiority for tall and skinny matrices ☺
- Future work: rectangular tiles
- Future work: robustness to variations in CPU performance

Conclusion

- Design of new tiled algorithms
- Exact or asymptotic formulas for their critical paths
- SimGrid simulator helpful!
- Experiments prove superiority for tall and skinny matrices ☺
- Future work: rectangular tiles
- Future work: robustness to variations in CPU performance