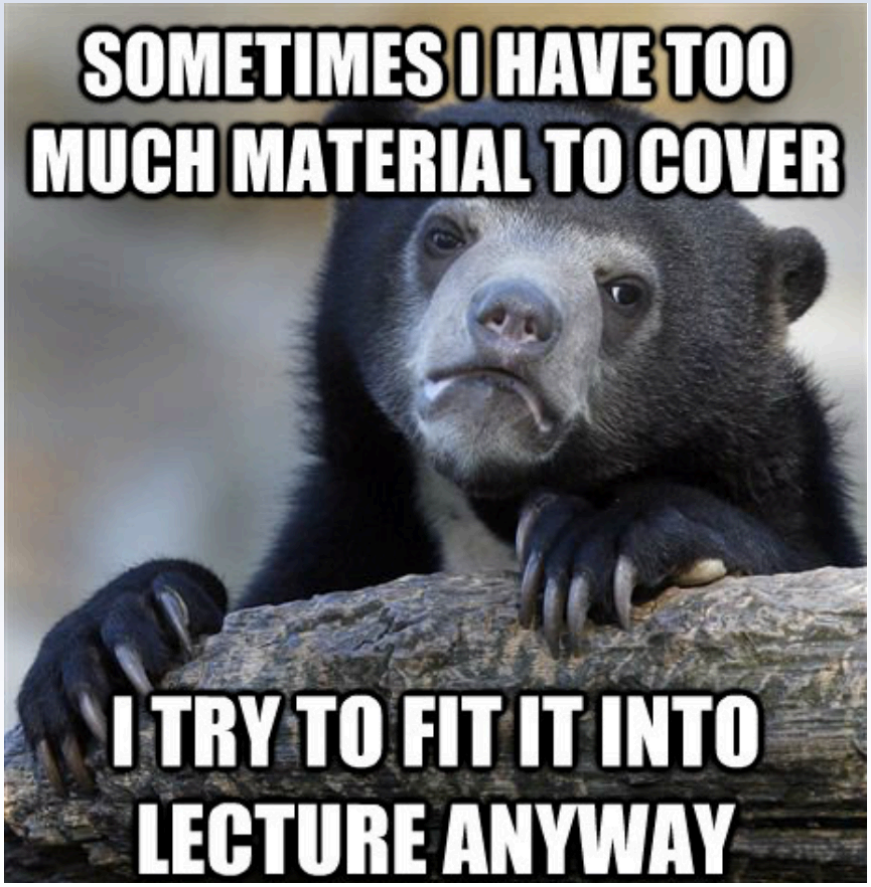
UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

CS411 - Logging and Recovery

illinois.edu

Mea Culpa


SOMETIMES I HAVE TOO MUCH MATERIAL TO COVER
I TRY TO FIT IT INTO LECTURE ANYWAY

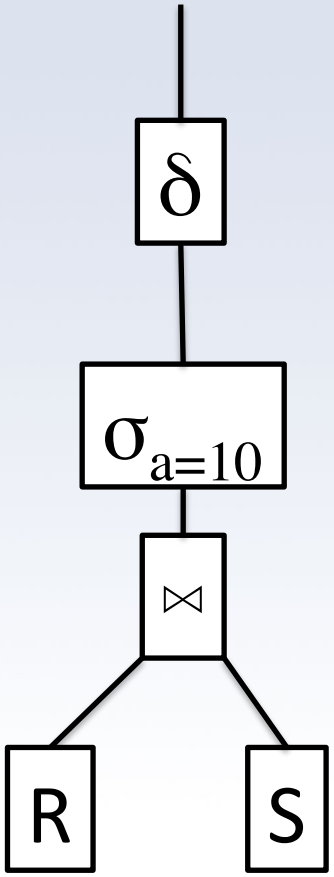
Cost Estimation

- ***How*** we do it: not that important
  - Can obviously do better
  - Estimation equations provided for exam
- That we ***can*** do it is important

Example

```
SELECT DISTINCT *
FROM R,S
WHERE R.a=10
```

$\delta(\sigma_{a=10}(R \bowtie S))$



Example

Given R(a,b) and S(b,c)

T(R)=5000, V(R,a)=50, V(R,b)=100

T(S)=2000, V(S,b)=200, V(S,c)=100

$U=\delta(\sigma_{a=10}(R \bowtie S))$

$T(R \bowtie S)=5000*2000/200=50000$

$T(\sigma_{a=10}(R \bowtie S))=50000/50=1000$

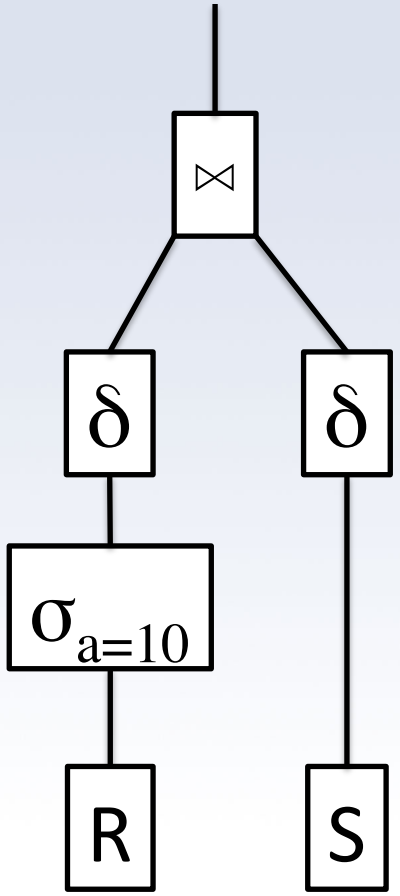
cost=51,000

T(U)=min(100,5000)=1000

Example

```
SELECT DISTINCT *
FROM R,S
WHERE R.a=10
```

$$\delta(\sigma_{a=10}(R)) \bowtie \delta(S)$$



Example

Given R(a,b) and S(b,c)

T(R)=5000, V(R,a)=50, V(R,b)=100

T(S)=2000, V(S,b)=200, V(S,c)=100

$U=\delta(\sigma_{a=10}(R))\bowtie\delta(S)$

$T(\sigma_{a=10}(R))=100$, $T(\delta(\sigma_{a=10}(R)))=50$

$T(\delta(S))=1000$

} cost=1,150

T(U)=1000*50/200=250

Example

```
SELECT a,b,AVG(x)
FROM R,S,T,U,V,W
WHERE C
GROUP BY a,b
```

Naïve Approach

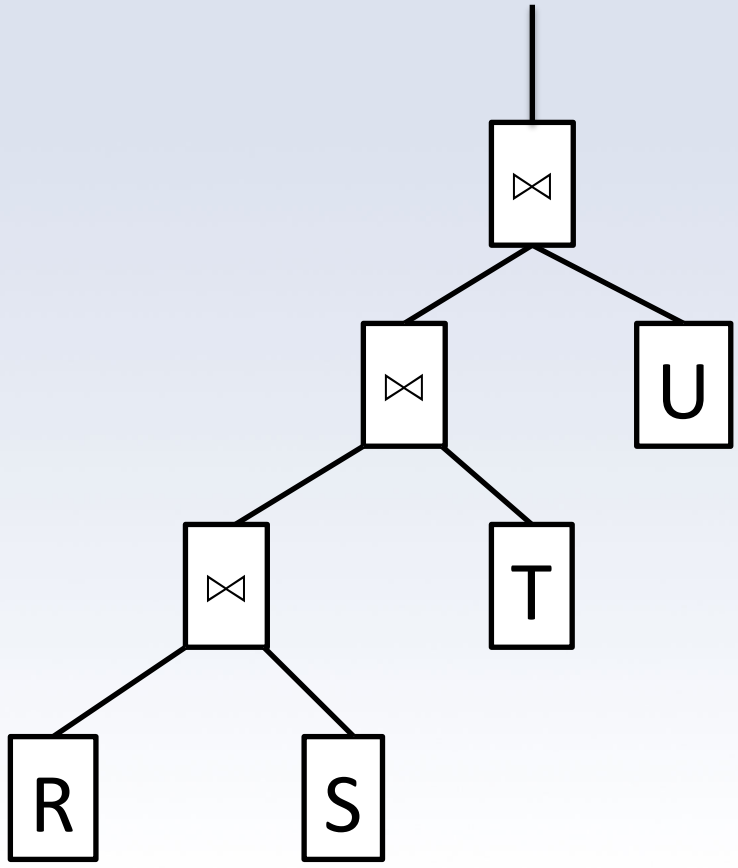
$\gamma_{a,b,avg(x)}(\sigma_C(R \bowtie S \bowtie T \bowtie U \bowtie V \bowtie W))$

- Number of binary trees with n leaves=(n-1) Catalan number= [2(n-1)]!/[(n!)(n-1)!]
- Number of arrangements of n leaves=n!
- Number of ways to reorder n joins=[2(n-1)]!/(n-1)!

Reordering JUST the joins in this problem: 30,240

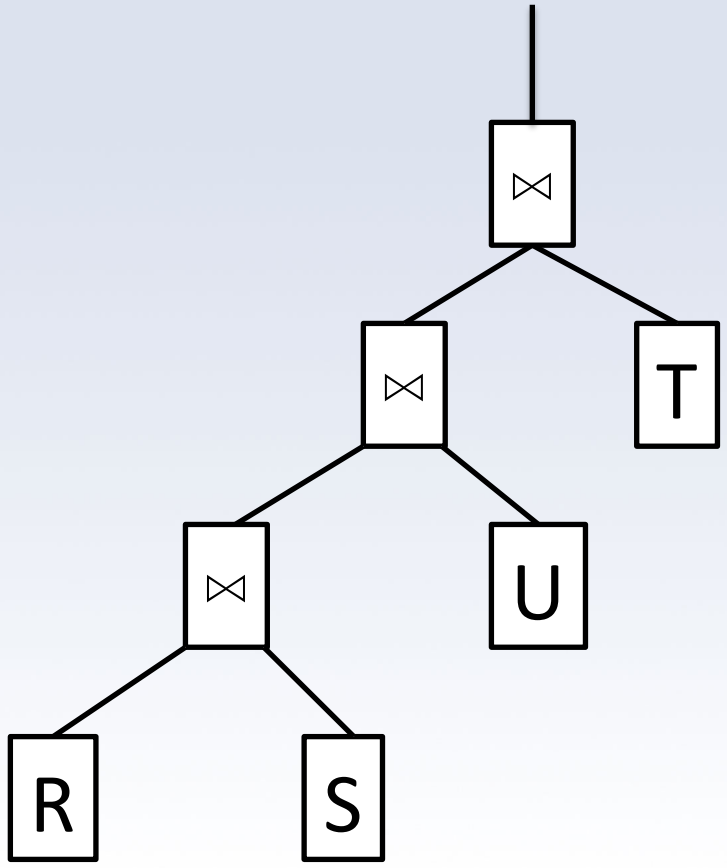
For 10 joins: 17,643,225,600

Left Join Tree



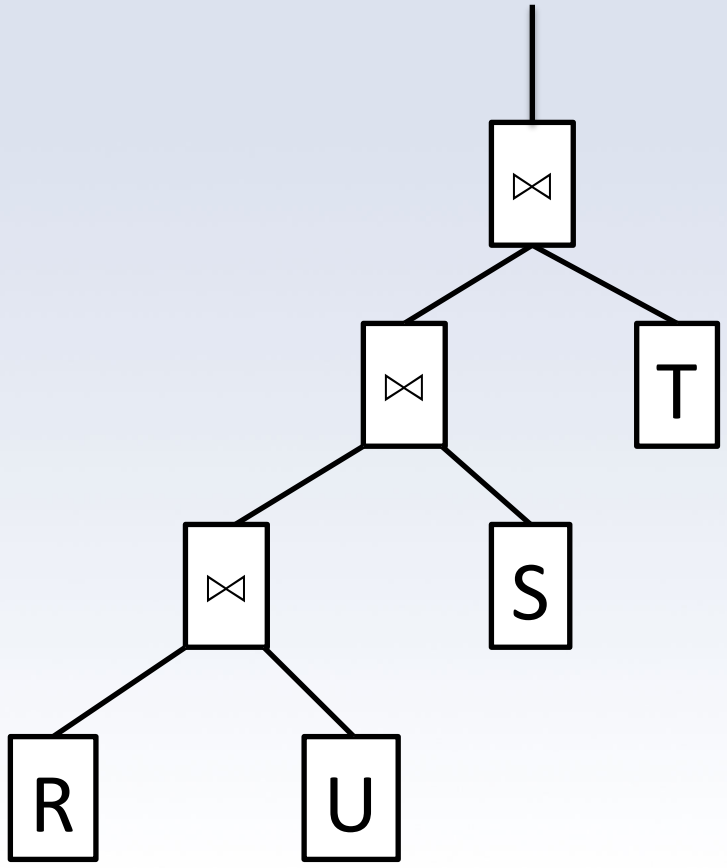
- A tree topology with two advantages:
  1. Only n! different orderings possible
  2. Can be easily pipelined with our algorithms
- On examples/exams, only use left join trees

Left Join Tree



- A tree topology with two advantages:
  1. Only n! different orderings possible
  2. Can be easily pipelined with our algorithms
- On examples/exams, only use left join trees

Left Join Tree



- A tree topology with two advantages:
  1. Only n! different orderings possible
  2. Can be easily pipelined with our algorithms
- On examples/exams, only use left join trees

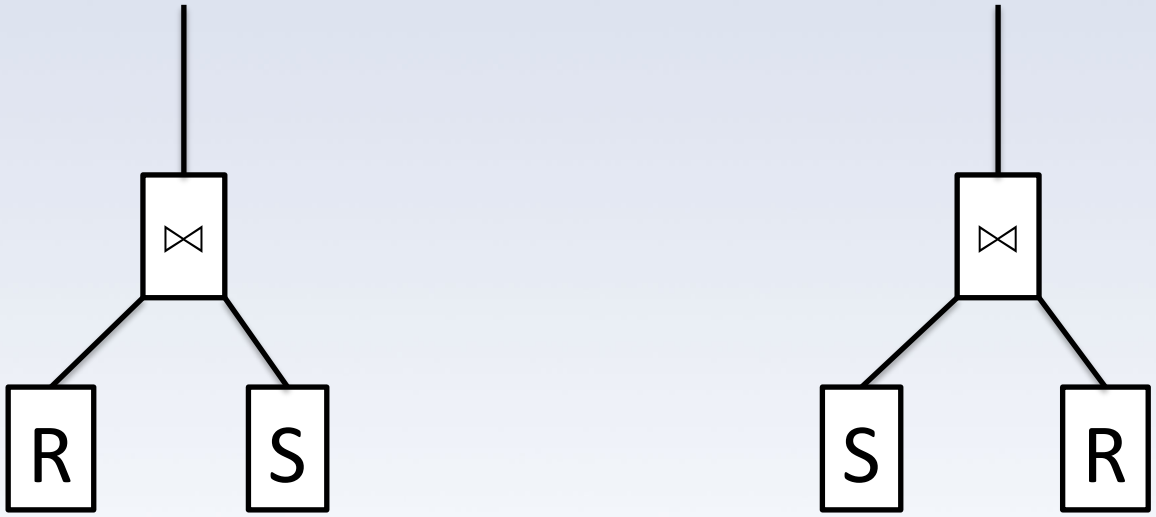
Dynamic Programming

- Main idea: in large combinatorial problems, many of the partial results are helpful in later steps
- Example:
  Cost((R⋈S)⋈T) result is useful for computing
  Cost((R⋈S)⋈T)⋈U)
  Cost(((R⋈S)⋈T)⋈V)
  Cost(((R⋈S)⋈T)⋈W)

Example

Subset	Size(S)	Cost	Plan
{R,S}			
{R,T}			
{R,U}			
{S,T}			
{S,U}			
{T,U}			
{R,S,T}			
{R,S,U}			
{R,T,U}			
{S,T,U}			
{R,S,T,U}			

Subsets of size 2



Example

Subset	Size(S)	Cost	Plan
{R,S}			R⋈S
{R,T}			
{R,U}			
{S,T}			
{S,U}			
{T,U}			
{R,S,T}			
{R,S,U}			
{R,T,U}			
{S,T,U}			
{R,S,T,U}			

Example

Subset	Size(S)	Cost	Plan
{R,S}	T(R)T(S)/max(V(R,b),V(S,b))		R⋈S
{R,T}			
{R,U}			
{S,T}			
{S,U}			
{T,U}			
{R,S,T}			
{R,S,U}			
{R,T,U}			
{S,T,U}			
{R,S,T,U}			

Example

Subset	Size(S)	Cost	Plan
{R,S}	1k*1k/200		R⋈S
{R,T}			
{R,U}			
{S,T}			
{S,U}			
{T,U}			
{R,S,T}			
{R,S,U}			
{R,T,U}			
{S,T,U}			
{R,S,T,U}			

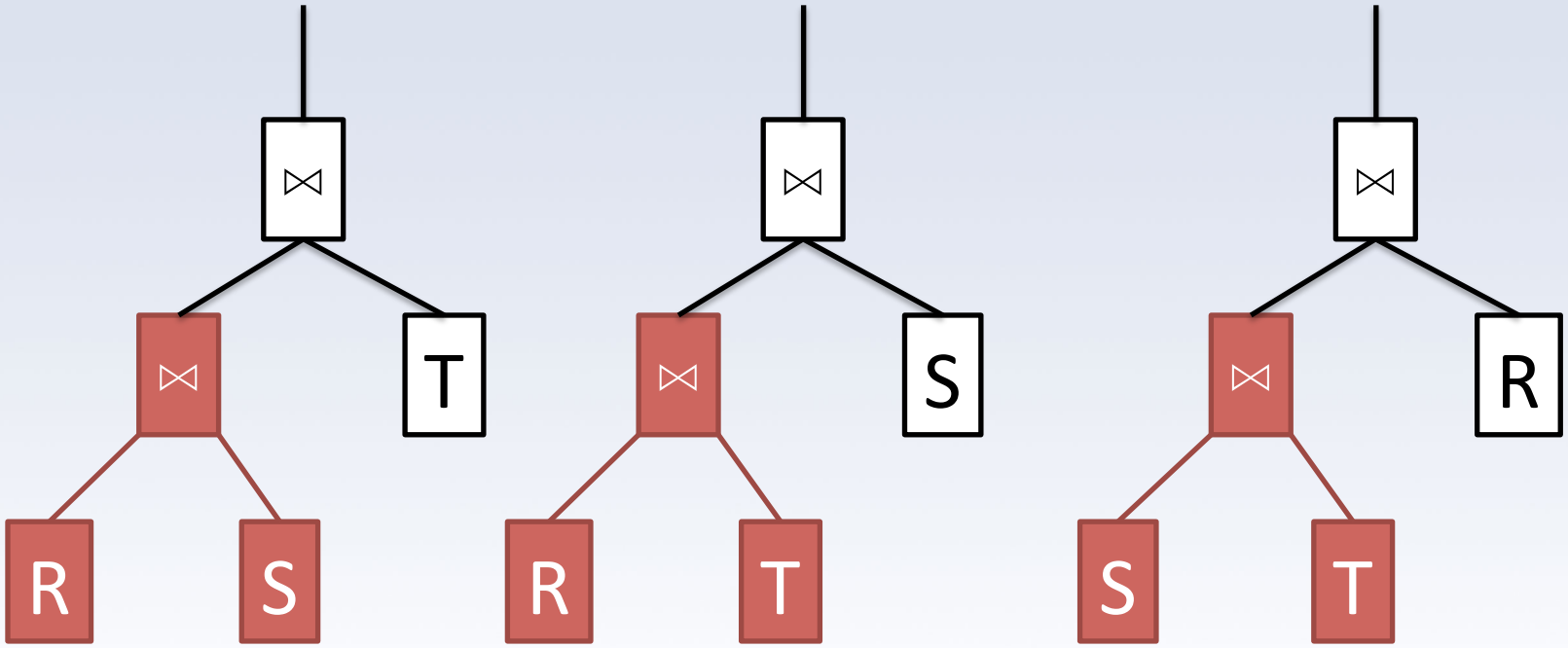
Example

Subset	Size(S)	Cost	Plan
{R,S}	5k	0	R⋈S
{R,T}			
{R,U}			
{S,T}			
{S,U}			
{T,U}			
{R,S,T}			
{R,S,U}			
{R,T,U}			
{S,T,U}			
{R,S,T,U}			

Example

Subset	Size(S)	Cost	Plan
{R,S}	5k	0	R⋈S
{R,T}	1,000k	0	R⋈T
{R,U}	10k	0	R⋈U
{S,T}	2k	0	S⋈T
{S,U}	1,000k	0	S⋈U
{T,U}	1k	0	T⋈U
{R,S,T}			
{R,S,U}			
{R,T,U}			
{S,T,U}			
{R,S,T,U}			

Subsets of size 3



Example

Subset	Size(S)	Cost	Plan
{R,S}	5k	0	R⋈S
{R,T}	1,000k	0	R⋈T
{R,U}	10k	0	R⋈U
{S,T}	2k	0	S⋈T
{S,U}	1,000k	0	S⋈U
{T,U}	1k	0	T⋈U
{R,S,T}	(1k*1k*1k)/(500*200)=10k	5k+0	(R⋈S)⋈T
{R,S,U}			
{R,T,U}			
{S,T,U}			
{R,S,T,U}			

Example

Subset	Size(S)	Cost	Plan
{R,S}	5k	0	R⋈S
{R,T}	1,000k	0	R⋈T
{R,U}	10k	0	R⋈U
{S,T}	2k	0	S⋈T
{S,U}	1,000k	0	S⋈U
{T,U}	1k	0	T⋈U
{R,S,T}	(1k*1k*1k)/(500*200)=10k	1,000k+0	(R⋈T)⋈S
{R,S,U}			
{R,T,U}			
{S,T,U}			
{R,S,T,U}			

Example

Subset	Size(S)	Cost	Plan
{R,S}	5k	0	R⋈S
{R,T}	1,000k	0	R⋈T
{R,U}	10k	0	R⋈U
{S,T}	2k	0	S⋈T
{S,U}	1,000k	0	S⋈U
{T,U}	1k	0	T⋈U
{R,S,T}	(1k*1k*1k)/(500*200)=10k	2k+0	(S⋈T)⋈R
{R,S,U}			
{R,T,U}			
{S,T,U}			
{R,S,T,U}			

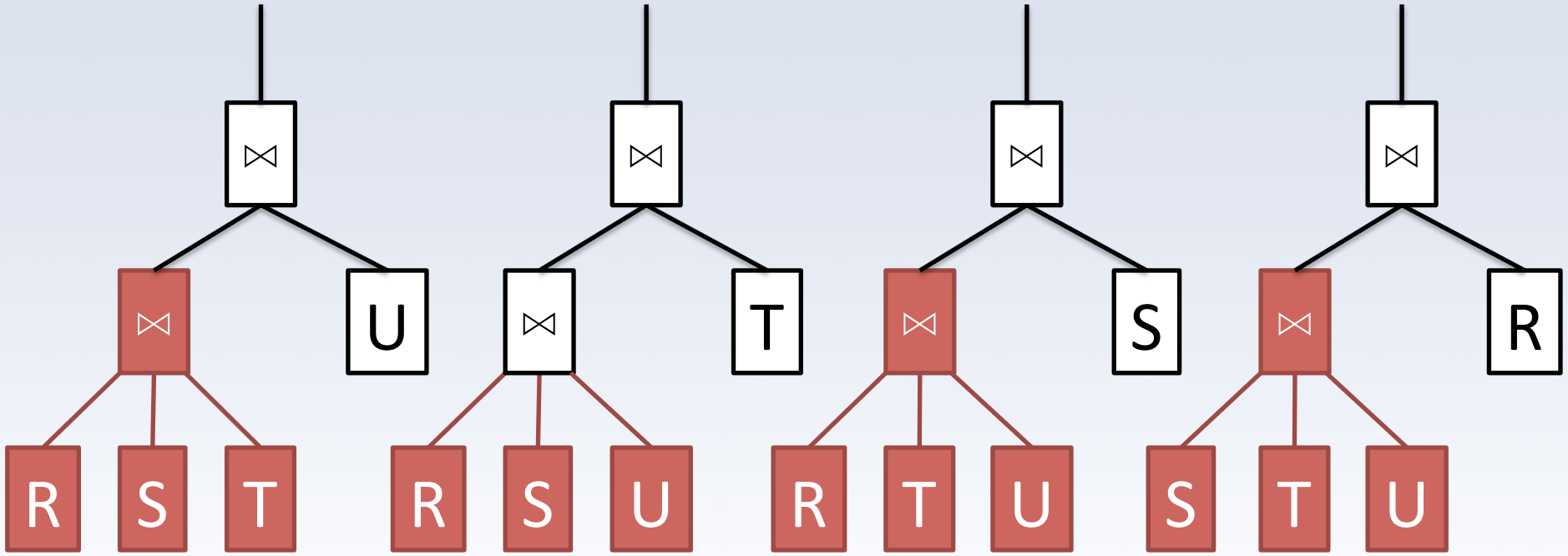
Example

Subset	Size(S)	Cost	Plan
{R,S}	5k	0	R⋈S
{R,T}	1,000k	0	R⋈T
{R,U}	10k	0	R⋈U
{S,T}	2k	0	S⋈T
{S,U}	1,000k	0	S⋈U
{T,U}	1k	0	T⋈U
{R,S,T}	10k	2k	(S⋈T)⋈R
{R,S,U}			
{R,T,U}			
{S,T,U}			
{R,S,T,U}			

Example

Subset	Size(S)	Cost	Plan
{R,S}	5k	0	R⋈S
{R,T}	1,000k	0	R⋈T
{R,U}	10k	0	R⋈U
{S,T}	2k	0	S⋈T
{S,U}	1,000k	0	S⋈U
{T,U}	1k	0	T⋈U
{R,S,T}	10k	2k	(S⋈T)⋈R
{R,S,U}	50k	5k	(R⋈S)⋈U
{R,T,U}	10k	1k	(T⋈U)⋈R
{S,T,U}	2k	1k	(T⋈U)⋈S
{R,S,T,U}			

Subsets of size 4



Example

Subset	Size(S)	Cost	Plan
{R,S}	5k	0	R⋈S
{R,T}	1,000k	0	R⋈T
{R,U}	10k	0	R⋈U
{S,T}	2k	0	S⋈T
{S,U}	1,000k	0	S⋈U
{T,U}	1k	0	T⋈U
{R,S,T}	10k	2k	(S⋈T)⋈R
{R,S,U}	50k	5k	(R⋈S)⋈U
{R,T,U}	10k	1k	(T⋈U)⋈R
{S,T,U}	2k	1k	(T⋈U)⋈S
{R,S,T,U}	(1k*1k*1k*1k)/ (1k*500*200*100)=10k	10k+2k+0+0=12k	((S⋈T)⋈R)⋈U

Example

Subset	Size(S)	Cost	Plan
{R,S}	5k	0	R⋈S
{R,T}	1,000k	0	R⋈T
{R,U}	10k	0	R⋈U
{S,T}	2k	0	S⋈T
{S,U}	1,000k	0	S⋈U
{T,U}	1k	0	T⋈U
{R,S,T}	10k	2k	(S⋈T)⋈R
{R,S,U}	50k	5k	(R⋈S)⋈U
{R,T,U}	10k	1k	(T⋈U)⋈R
{S,T,U}	2k	1k	(T⋈U)⋈S
{R,S,T,U}	(1k*1k*1k*1k)/ (1k*500*200*100)=10k	50k+5k+0+0=55k	((R⋈S)⋈U)⋈T

Example

Subset	Size(S)	Cost	Plan
{R,S}	5k	0	R⋈S
{R,T}	1,000k	0	R⋈T
{R,U}	10k	0	R⋈U
{S,T}	2k	0	S⋈T
{S,U}	1,000k	0	S⋈U
{T,U}	1k	0	T⋈U
{R,S,T}	10k	2k	(S⋈T)⋈R
{R,S,U}	50k	5k	(R⋈S)⋈U
{R,T,U}	10k	1k	(T⋈U)⋈R
{S,T,U}	2k	1k	(T⋈U)⋈S
{R,S,T,U}	(1k*1k*1k*1k)/ (1k*500*200*100)=10k	10k+1k+0+0=11k	((T⋈U)⋈R)⋈S

Example

Subset	Size(S)	Cost	Plan
{R,S}	5k	0	R⋈S
{R,T}	1,000k	0	R⋈T
{R,U}	10k	0	R⋈U
{S,T}	2k	0	S⋈T
{S,U}	1,000k	0	S⋈U
{T,U}	1k	0	T⋈U
{R,S,T}	10k	2k	(S⋈T)⋈R
{R,S,U}	50k	5k	(R⋈S)⋈U
{R,T,U}	10k	1k	(T⋈U)⋈R
{S,T,U}	2k	1k	(T⋈U)⋈S
{R,S,T,U}	(1k*1k*1k*1k)/ (1k*500*200*100)=10k	2k+1k+0+0=3k	((T⋈U)⋈S)⋈R

Example

Subset	Size(S)	Cost	Plan
{R,S}	5k	0	R⋈S
{R,T}	1,000k	0	R⋈T
{R,U}	10k	0	R⋈U
{S,T}	2k	0	S⋈T
{S,U}	1,000k	0	S⋈U
{T,U}	1k	0	T⋈U
{R,S,T}	10k	2k	(S⋈T)⋈R
{R,S,U}	50k	5k	(R⋈S)⋈U
{R,T,U}	10k	1k	(T⋈U)⋈R
{S,T,U}	2k	1k	(T⋈U)⋈S
{R,S,T,U}	10k	3k	((T⋈U)⋈S)⋈R

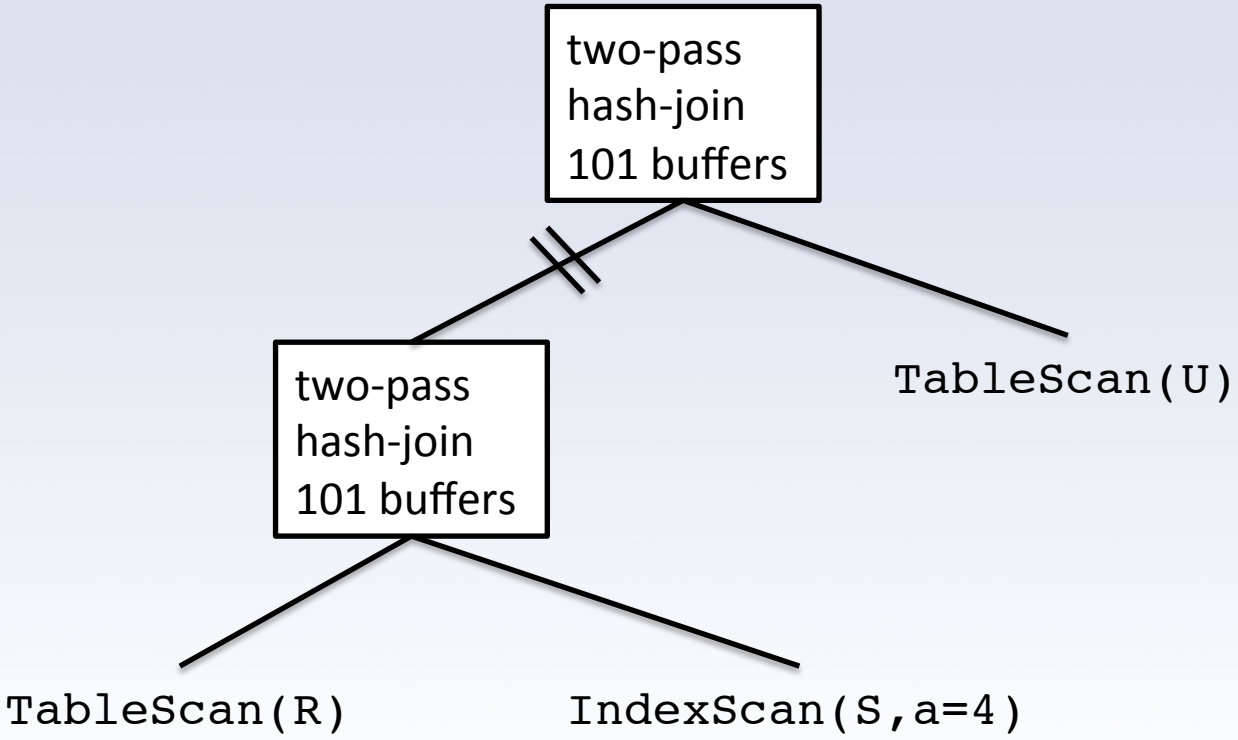
Physical Query Plan

- At each node of logical query plan:
  - Replace relational algebra operator
  - Substitute execution algorithms
    - How much memory is available?
    - Are relations indexed?
    - Are results sorted?
    - Should we pipeline or materialize intermediate results?

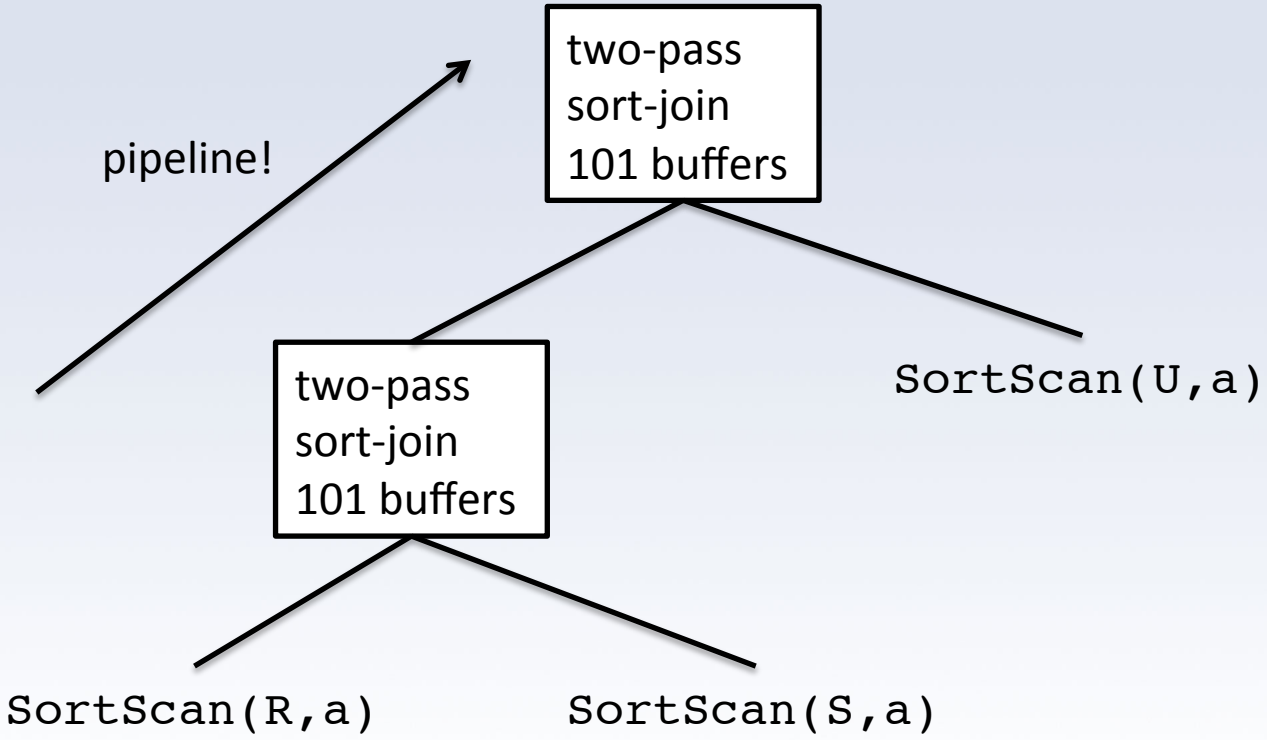
Pipelining vs. Materialization

- Materialization - write the results of the operation back to the disk
  - results in 2 intermediate disk I/Os per block
- Pipelining - keep results in memory buffers
  - Results of one iterator passed to the next

Physical Query Plan

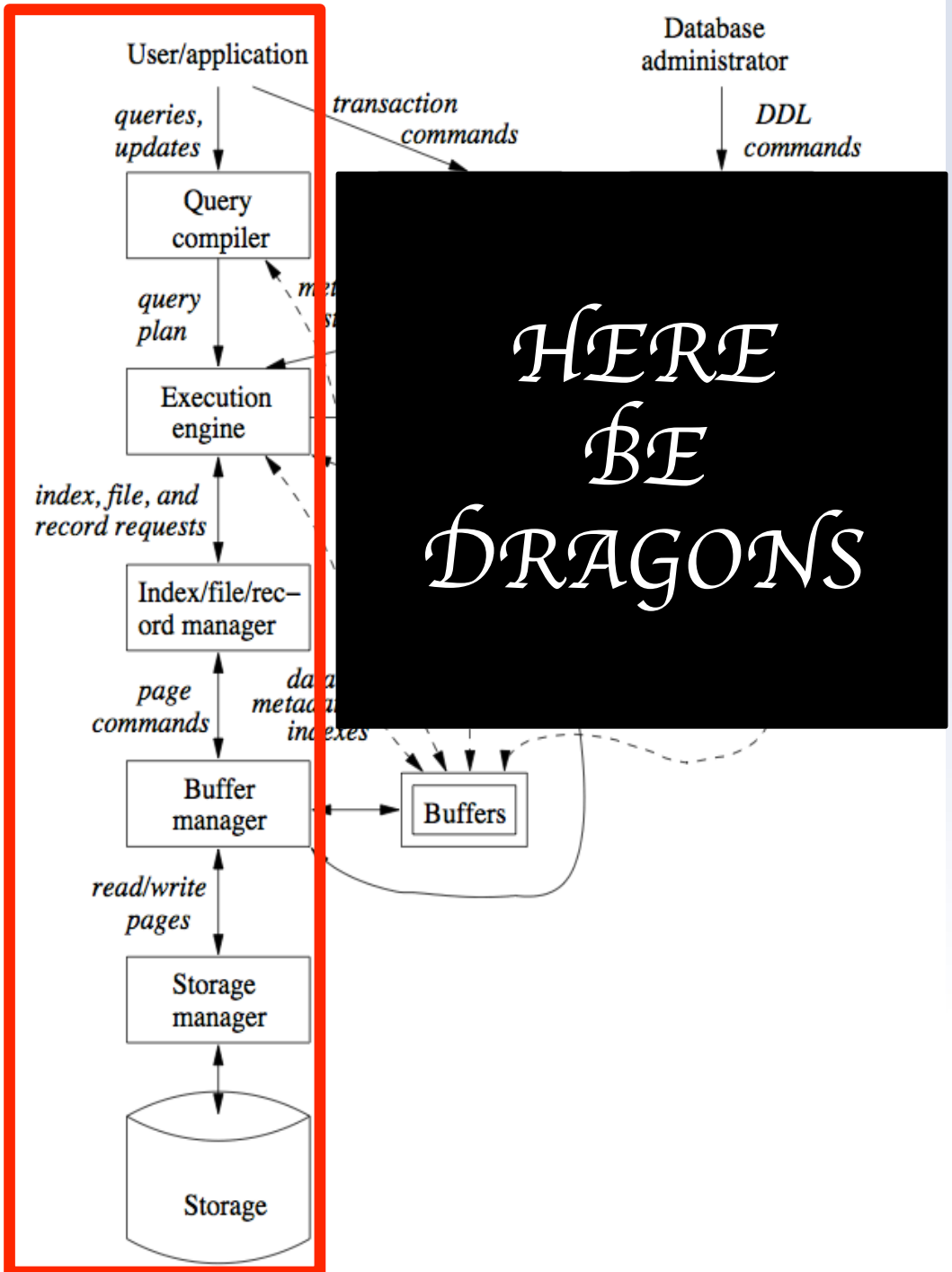


Physical Query Plan



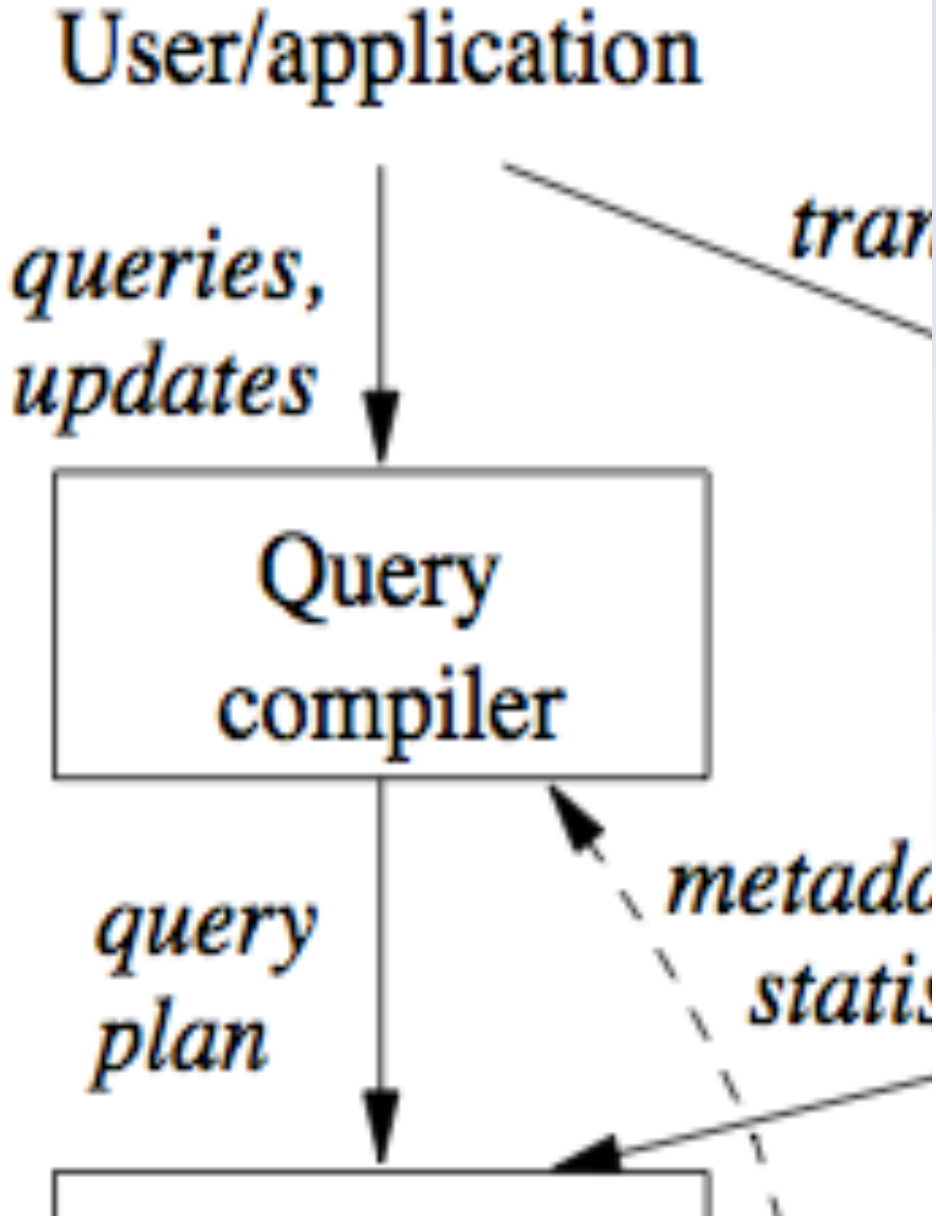
MP3

- We implemented a few algorithms for operators (e.g. select, join, project)
- We implemented cost estimates
- We have five queries
  - Improve the query plan for all five
  - Keeping within memory constraint


User/application
Database administrator
queries, updates
transaction commands
DDL commands
Query compiler
query plan
Execution engine
index, file, and record requests
Index/file/rec–ord manager
page commands
Buffer manager
Buffers
read/write pages
Storage manager
Storage
HERE BE DRAGONS

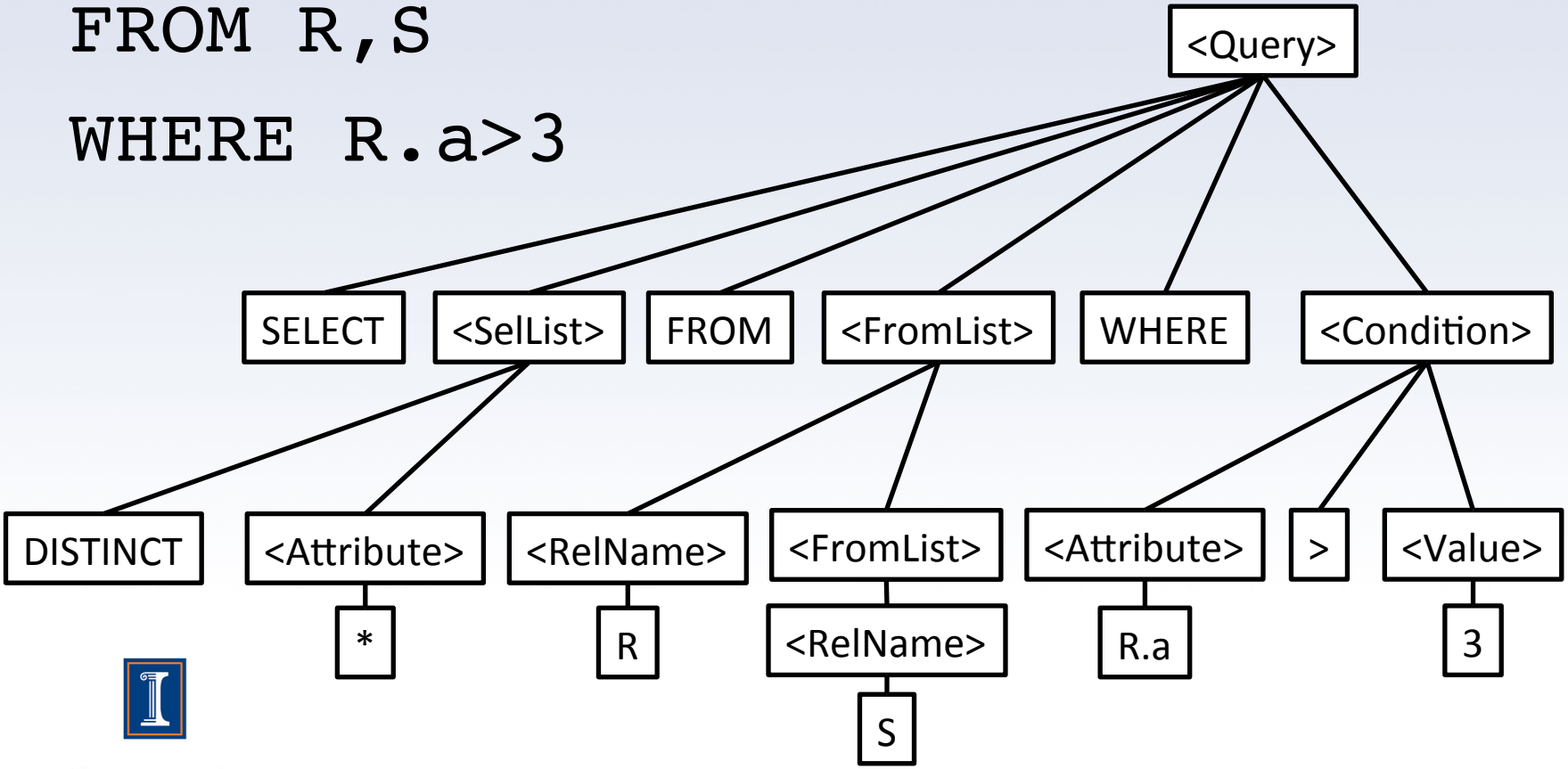
User Submits Query

```
SELECT DISTINCT *
FROM R,S
WHERE R.a>3
```

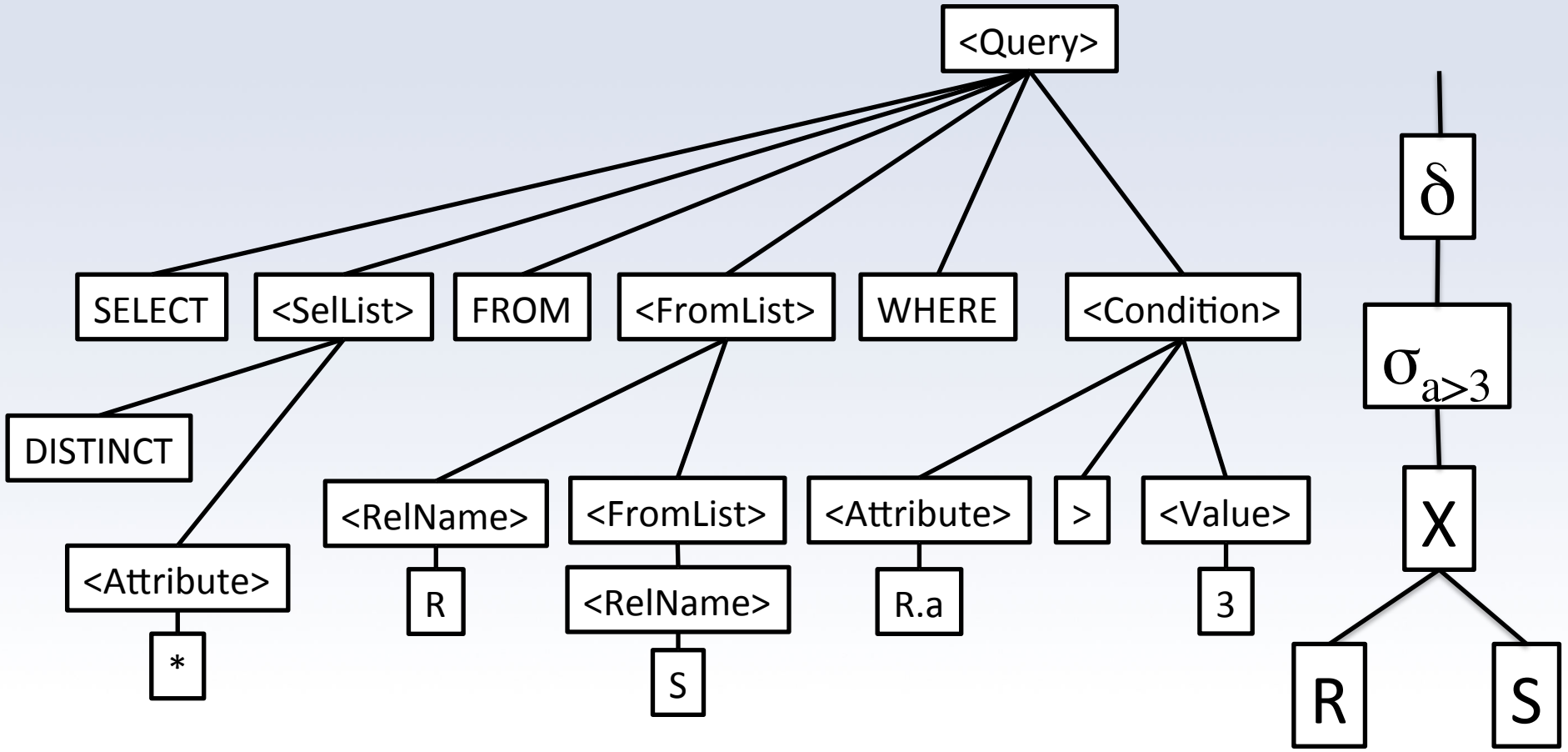

User/application
queries,
updates
Query
compiler
query
plan

Parse Tree

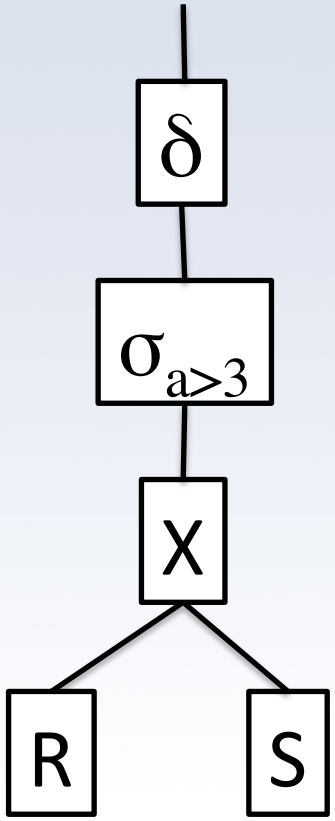
```
SELECT DISTINCT *
FROM R,S
WHERE R.a>3
```

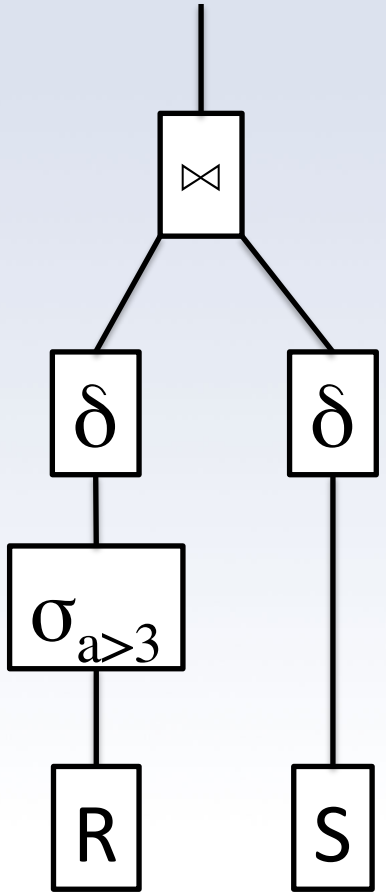


Logical Query Plan

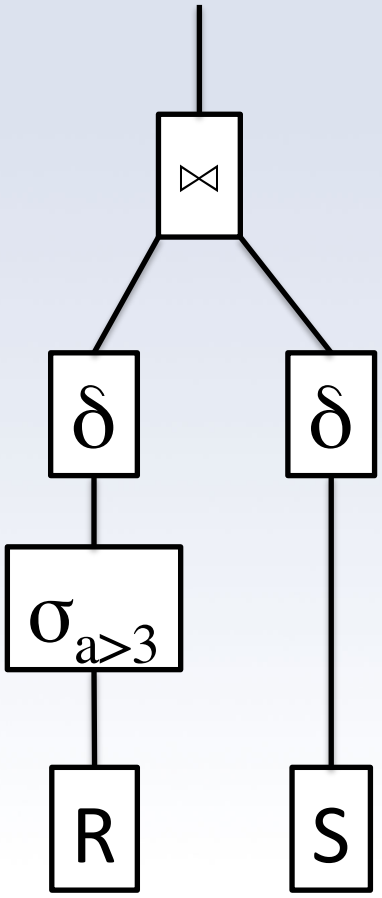


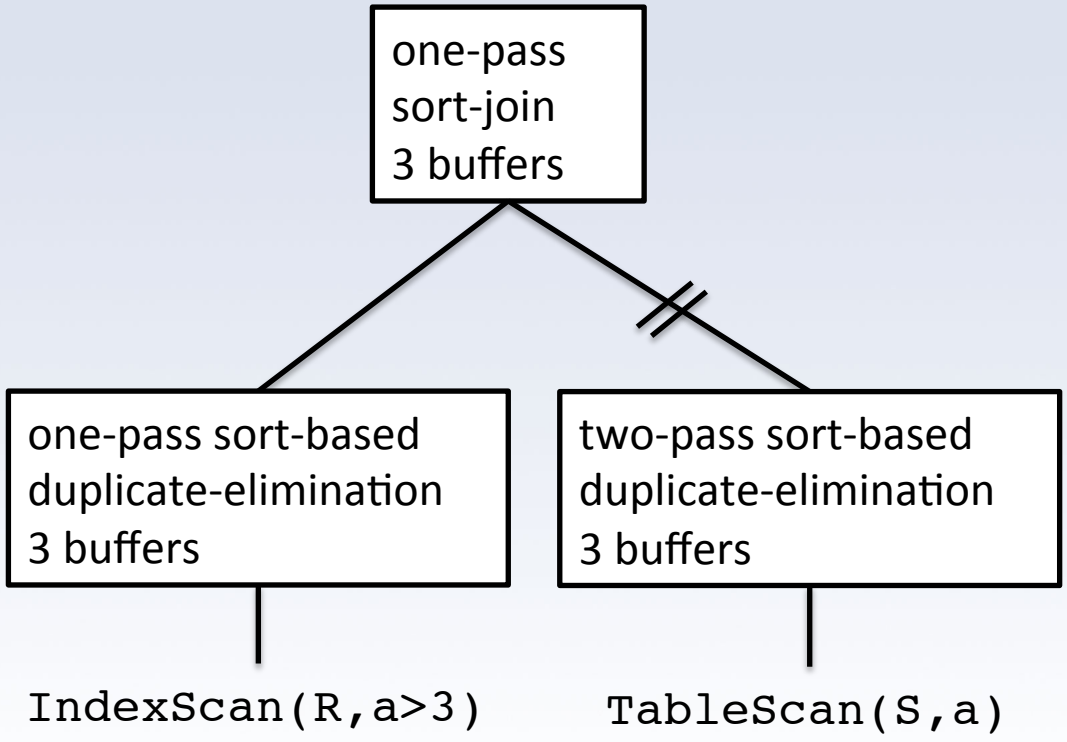
Logical Query Plan (Optimized)

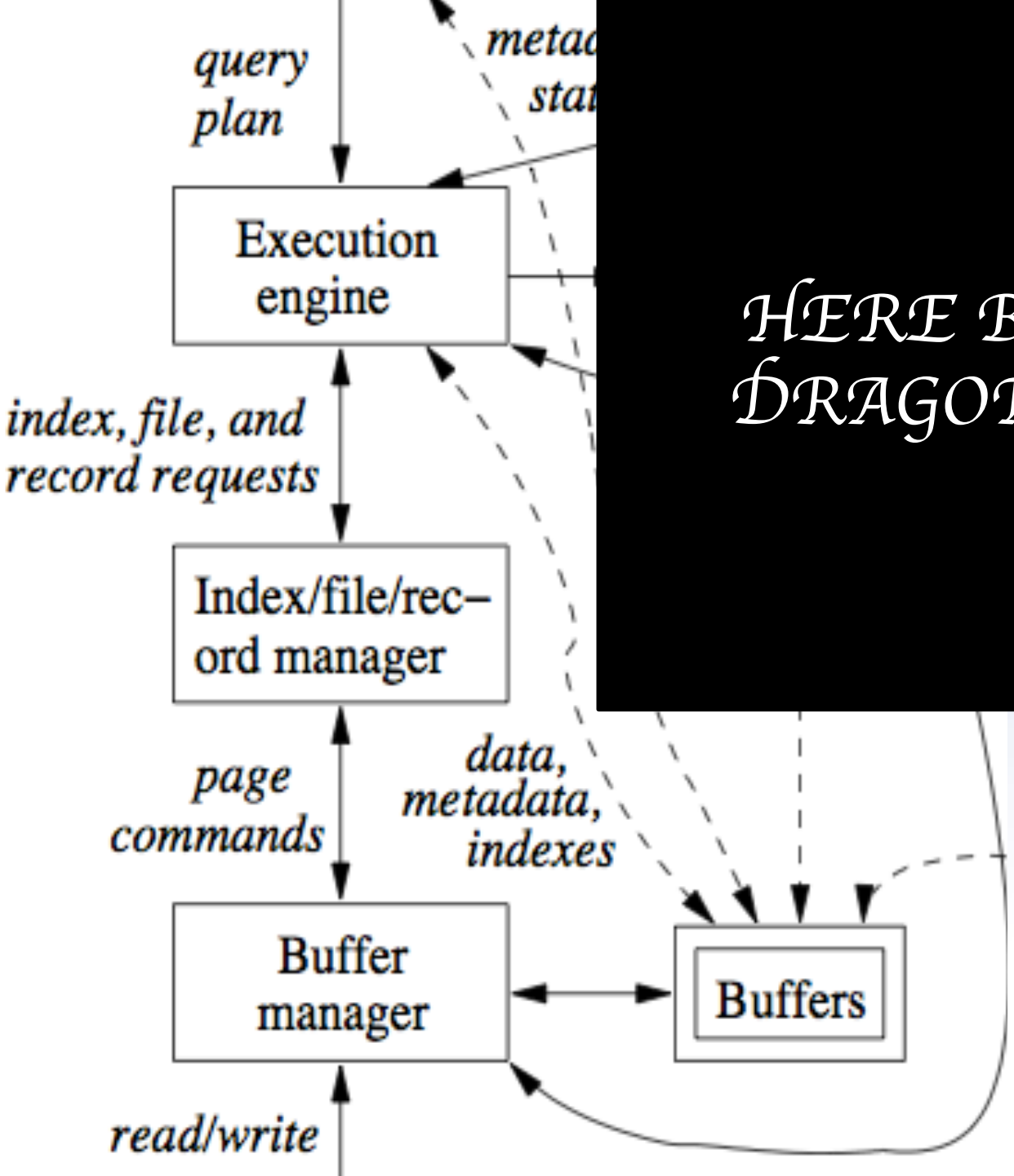




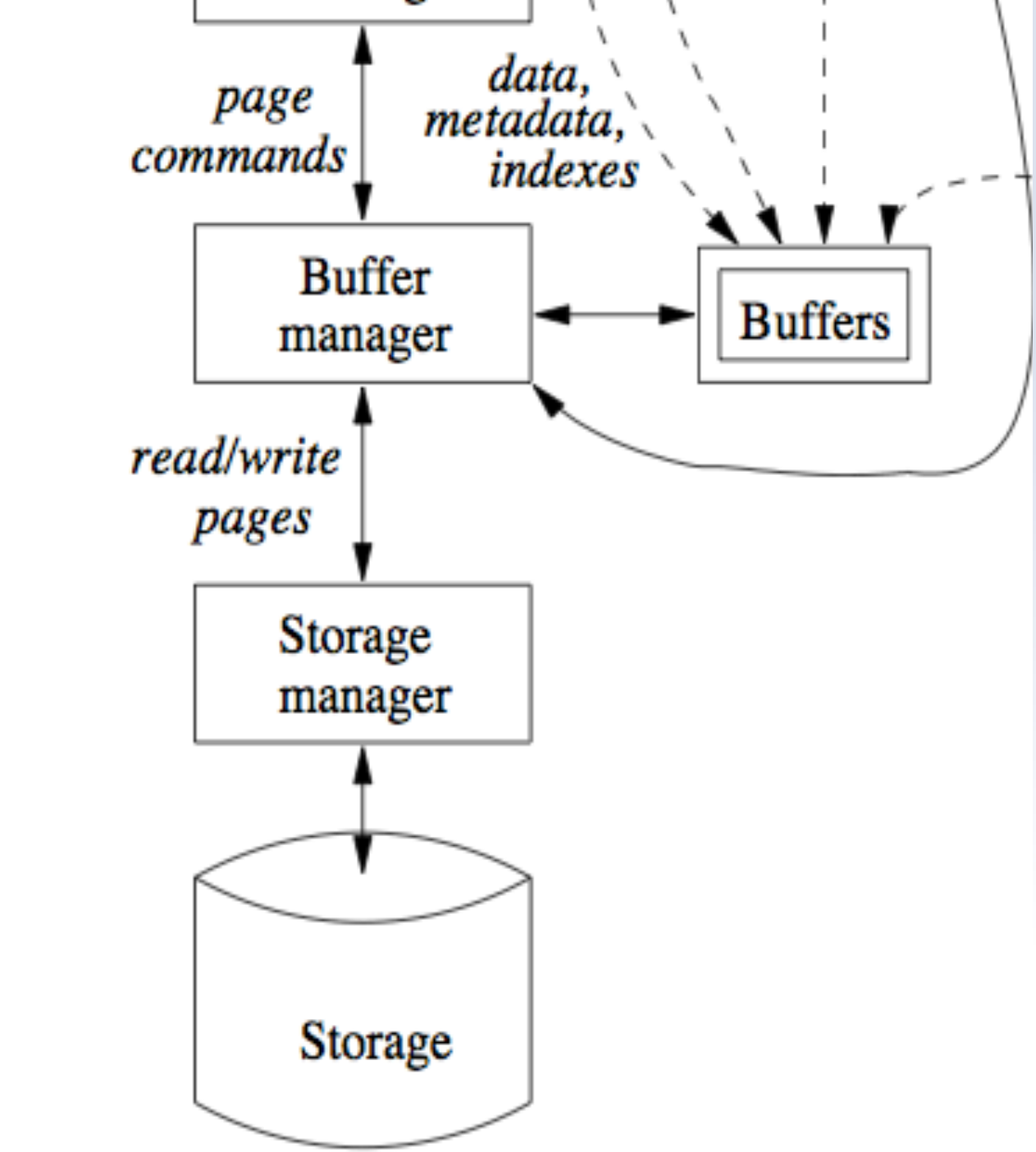
Physical Query Plan

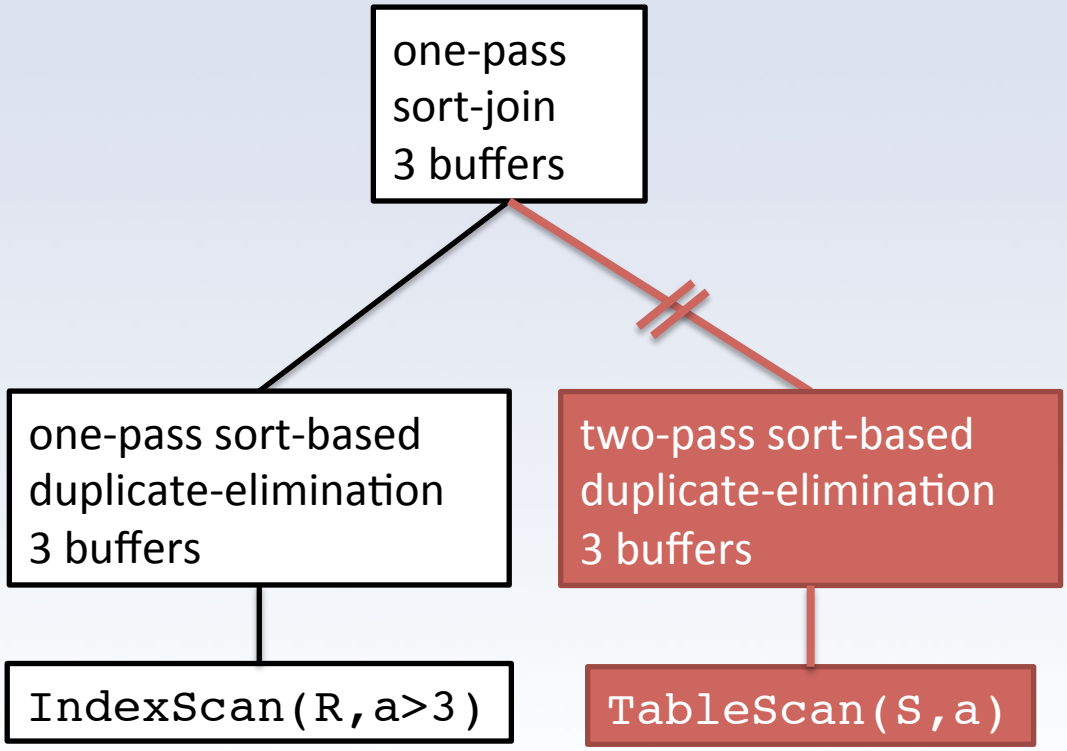




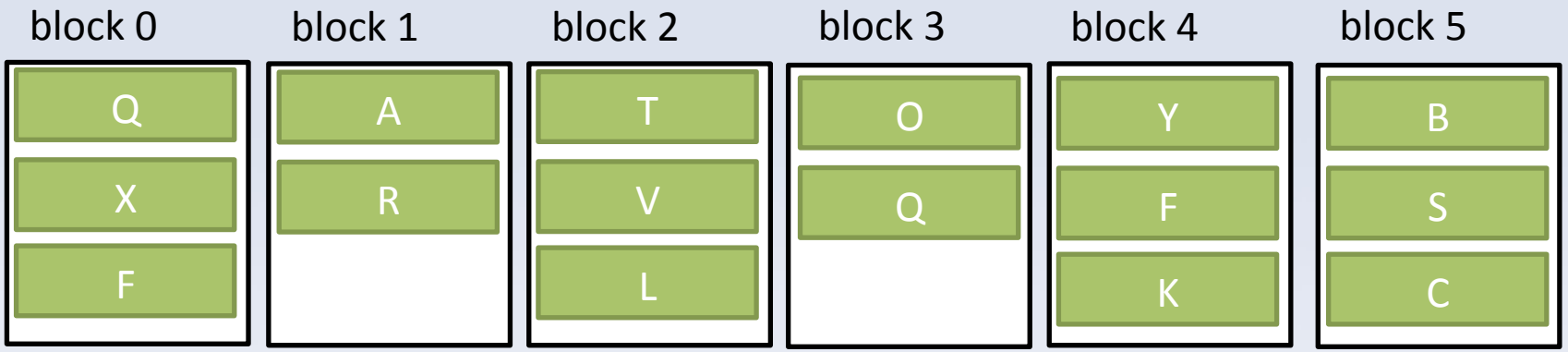


HERE BE DRAGONS

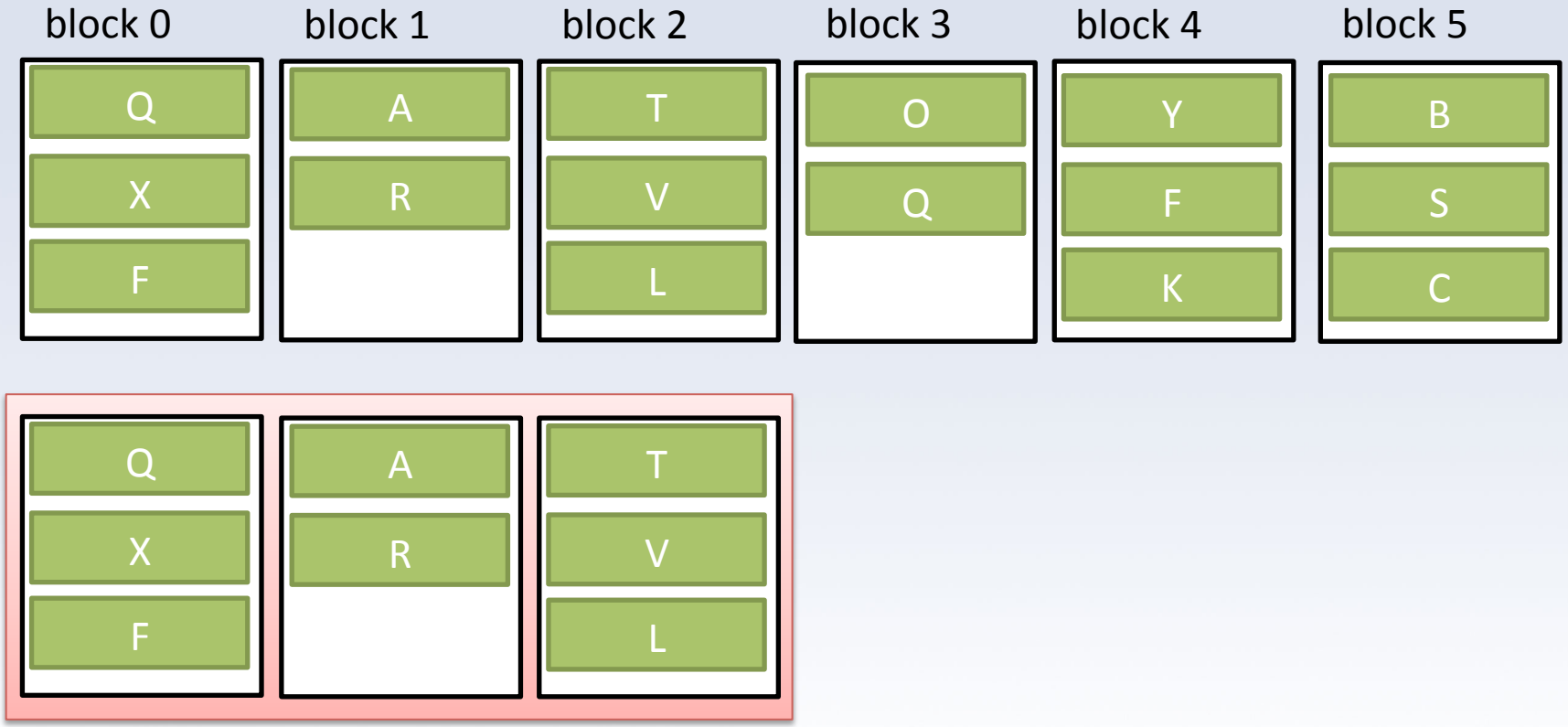

page
commands
data,
metadata,
indexes
Buffer
manager
Buffers
read/write
pages
Storage
manager
Storage


one-pass
sort-join
3 buffers
one-pass sort-based
duplicate-elimination
3 buffers
two-pass sort-based
duplicate-elimination
3 buffers
IndexScan(R,a>3)
TableScan(S,a)

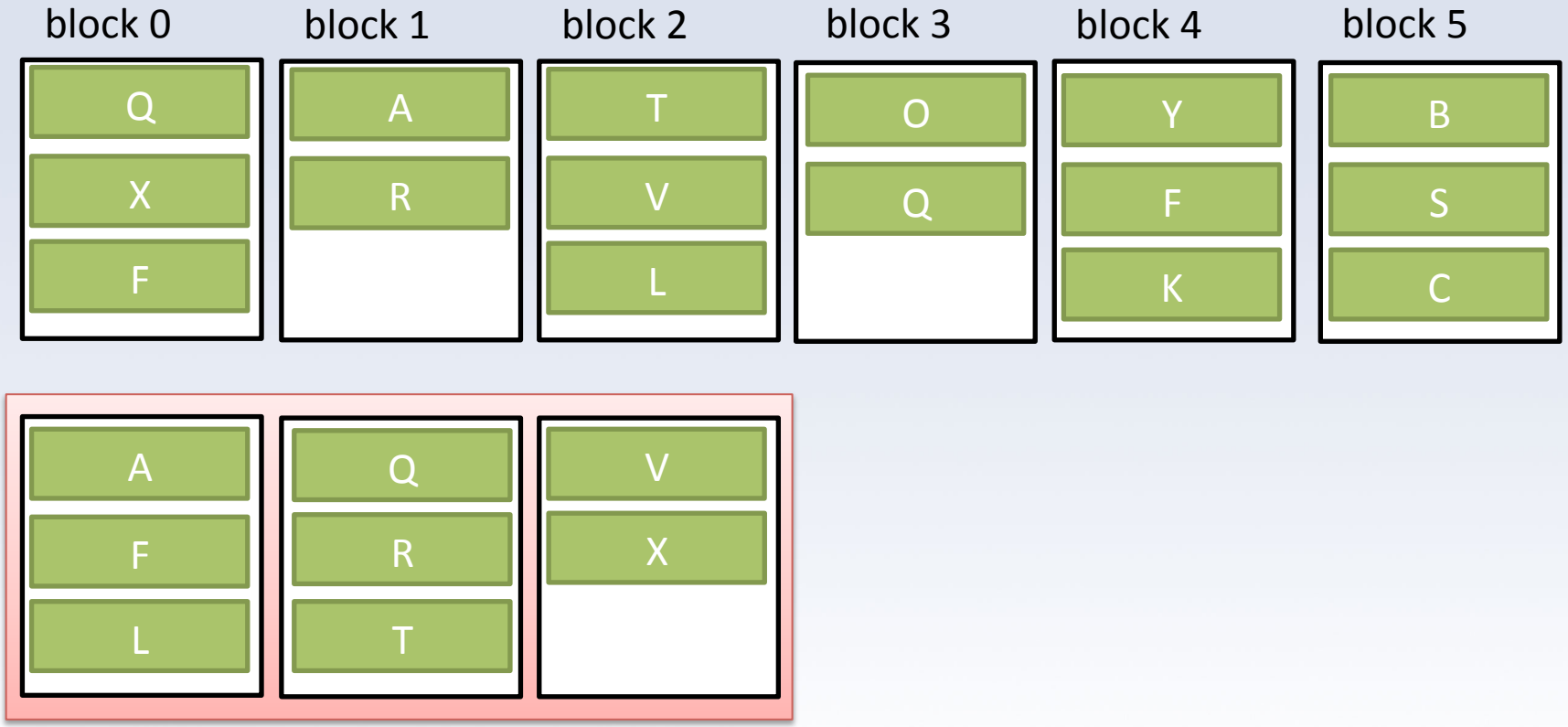
2 Pass Duplicate-Elimination



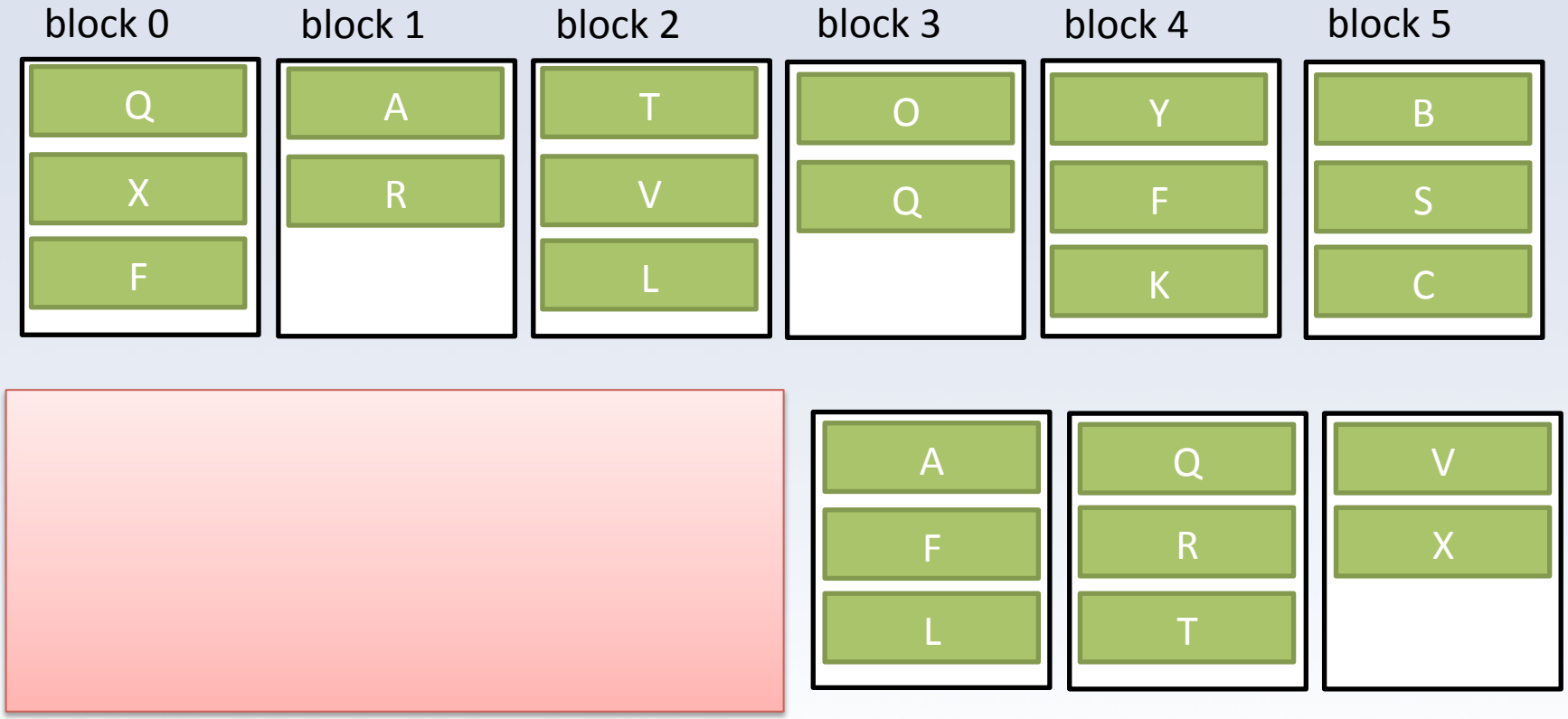
Phase 1


block 0
block 1
block 2
block 3
block 4
block 5
Q
X
F
A
R
T
V
L
O
Q
Y
F
K
B
S
C
Q
X
F
A
R
T
V
L

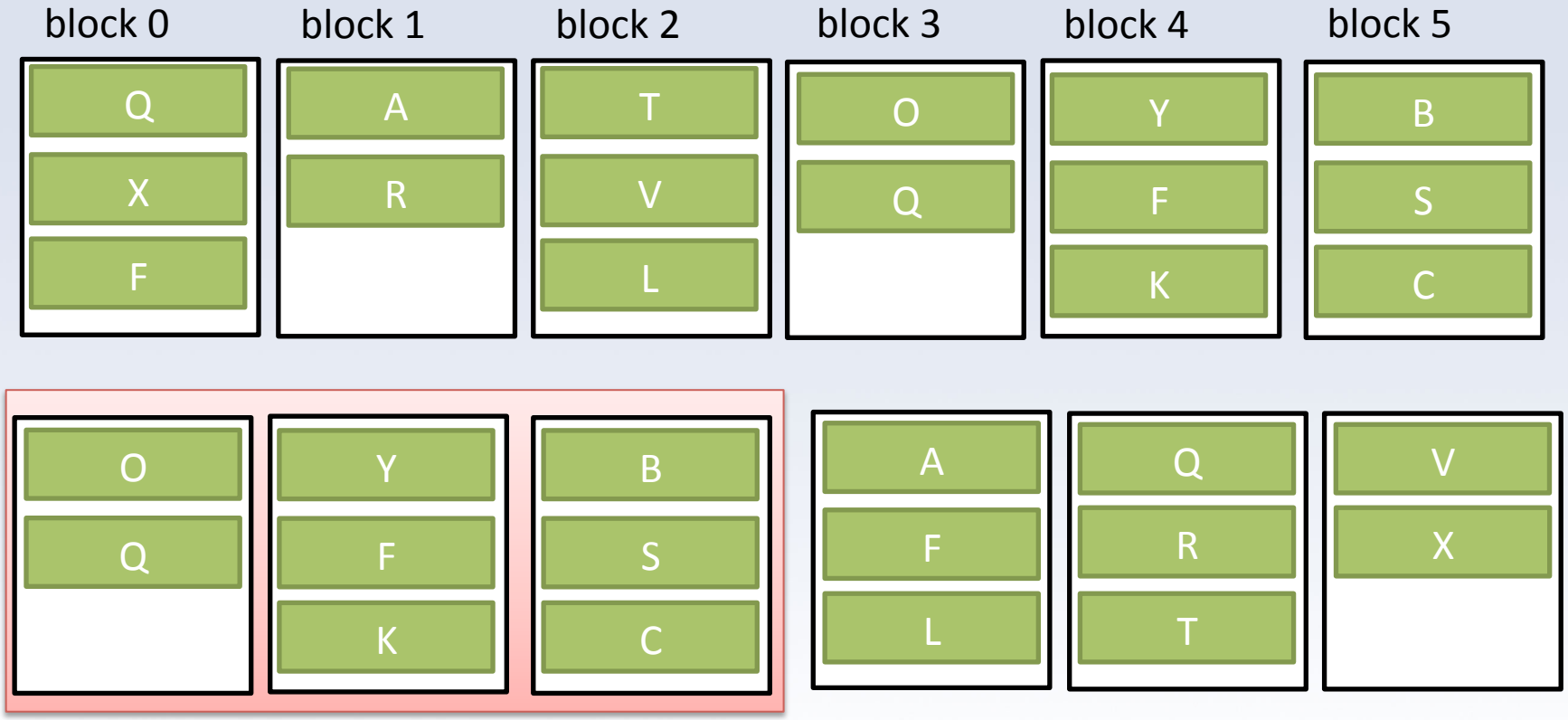
Phase 1


block 0
block 1
block 2
block 3
block 4
block 5
Q
X
F
A
R
T
V
L
O
Q
Y
F
K
B
S
C
A
F
L
Q
R
T
V
X

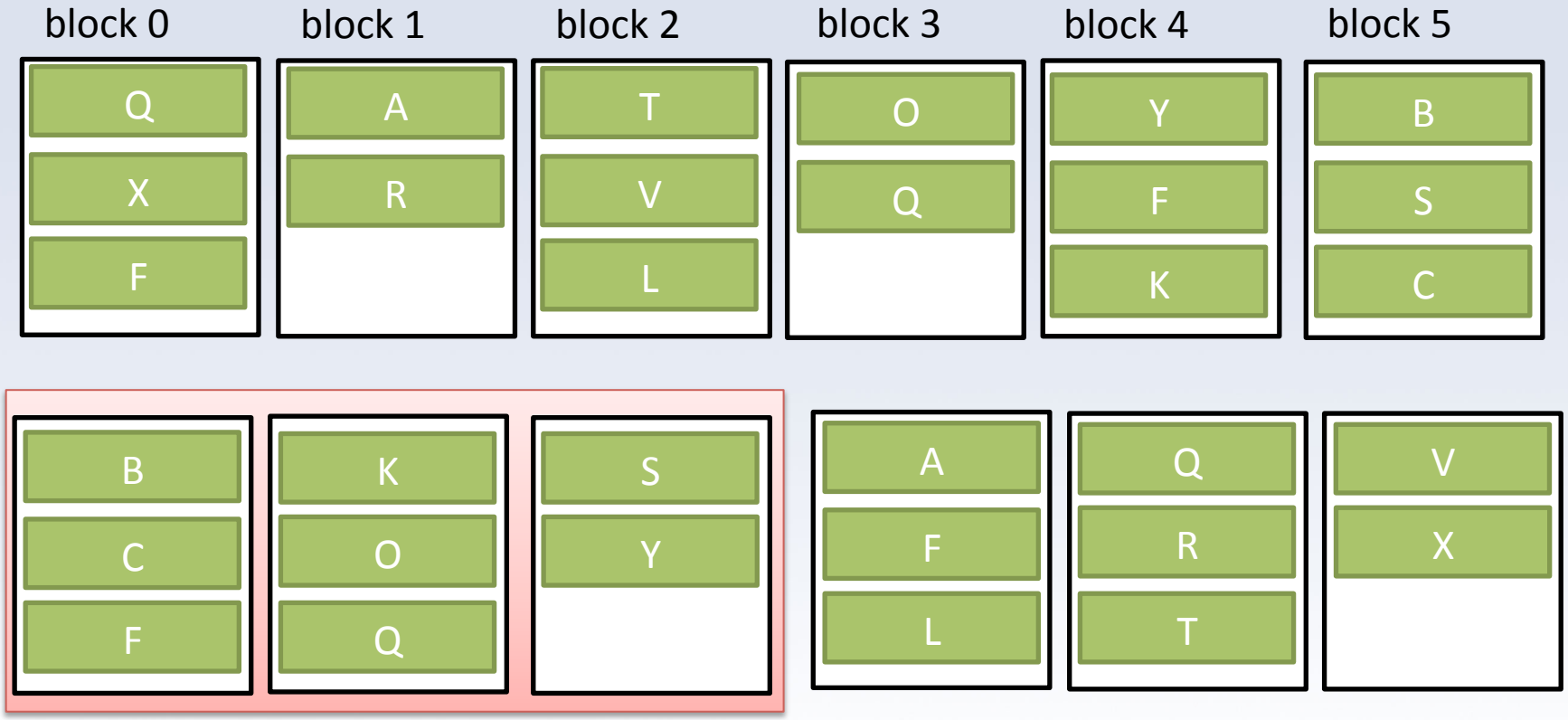
Phase 1


block 0
Q
X
F
block 1
A
R
block 2
T
V
L
block 3
O
Q
block 4
Y
F
K
block 5
B
S
C
A
F
L
Q
R
T
V
X

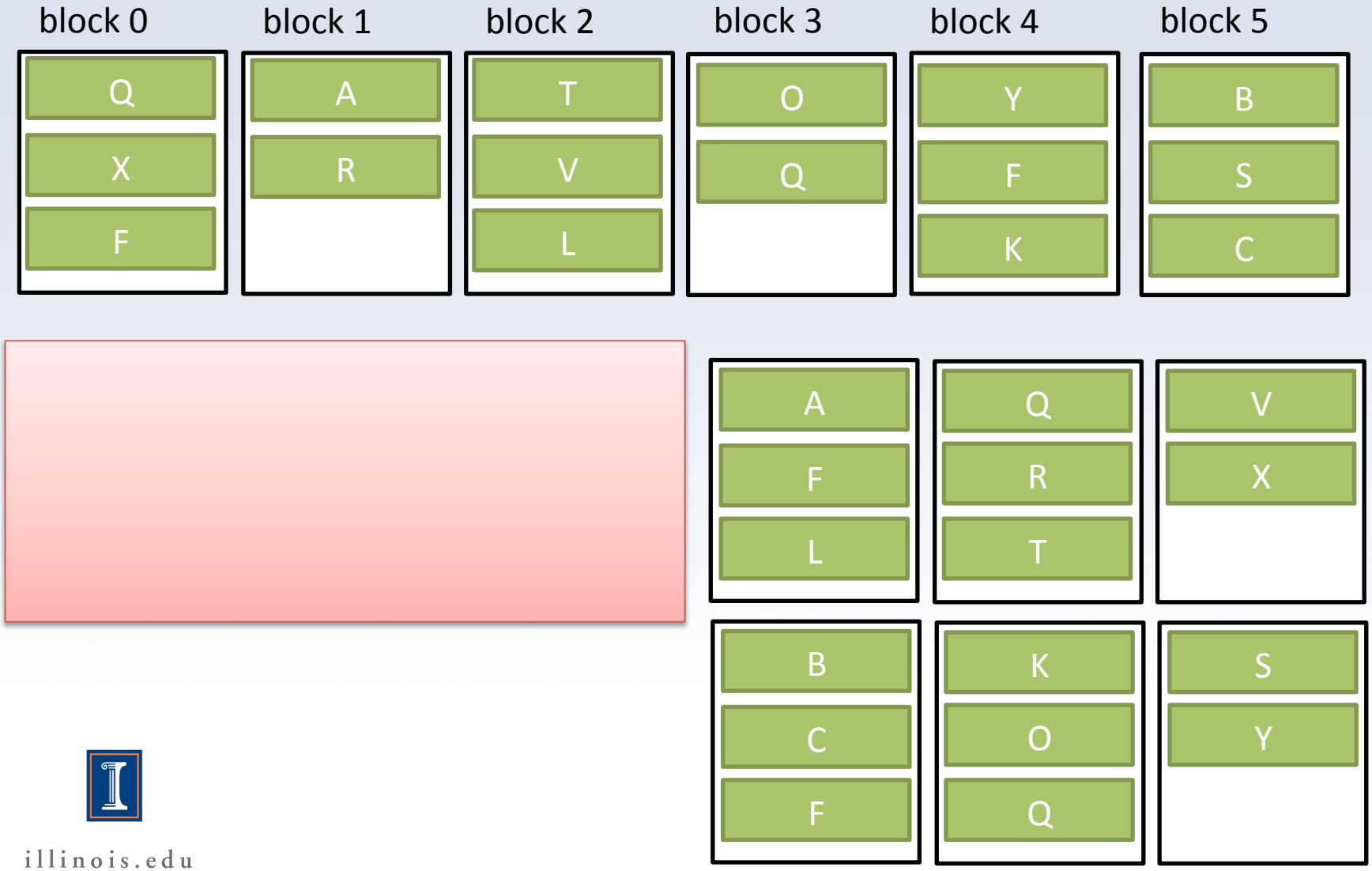
Phase 1


block 0
Q
X
F
block 1
A
R
block 2
T
V
L
block 3
O
Q
block 4
Y
F
K
block 5
B
S
C
O
Q
Y
F
K
B
S
C
A
F
L
Q
R
T
V
X

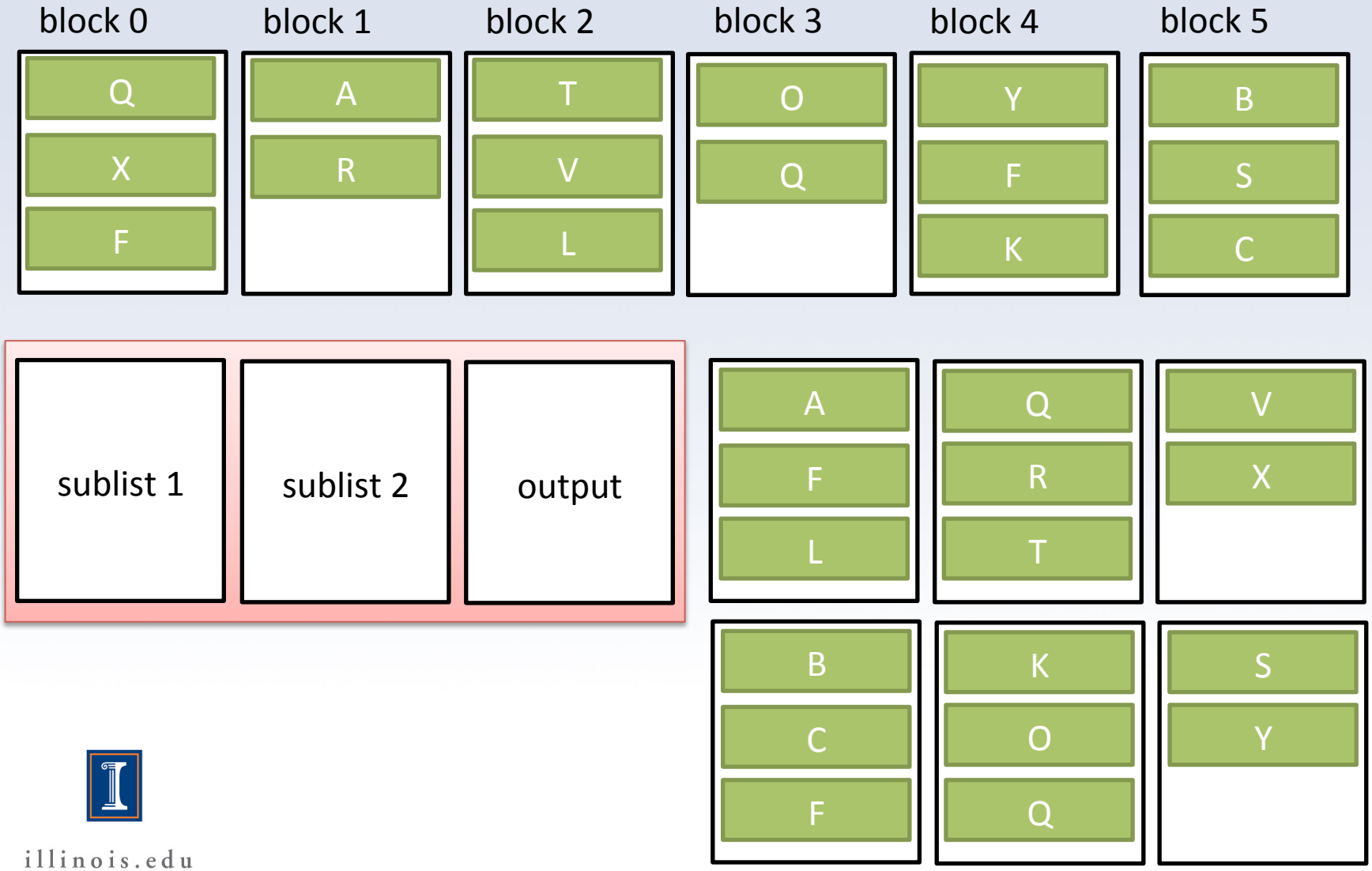
Phase 1


block 0
block 1
block 2
block 3
block 4
block 5
Q
X
F
A
R
T
V
L
O
Q
Y
F
K
B
S
C
B
C
F
K
O
Q
S
Y
A
F
L
Q
R
T
V
X

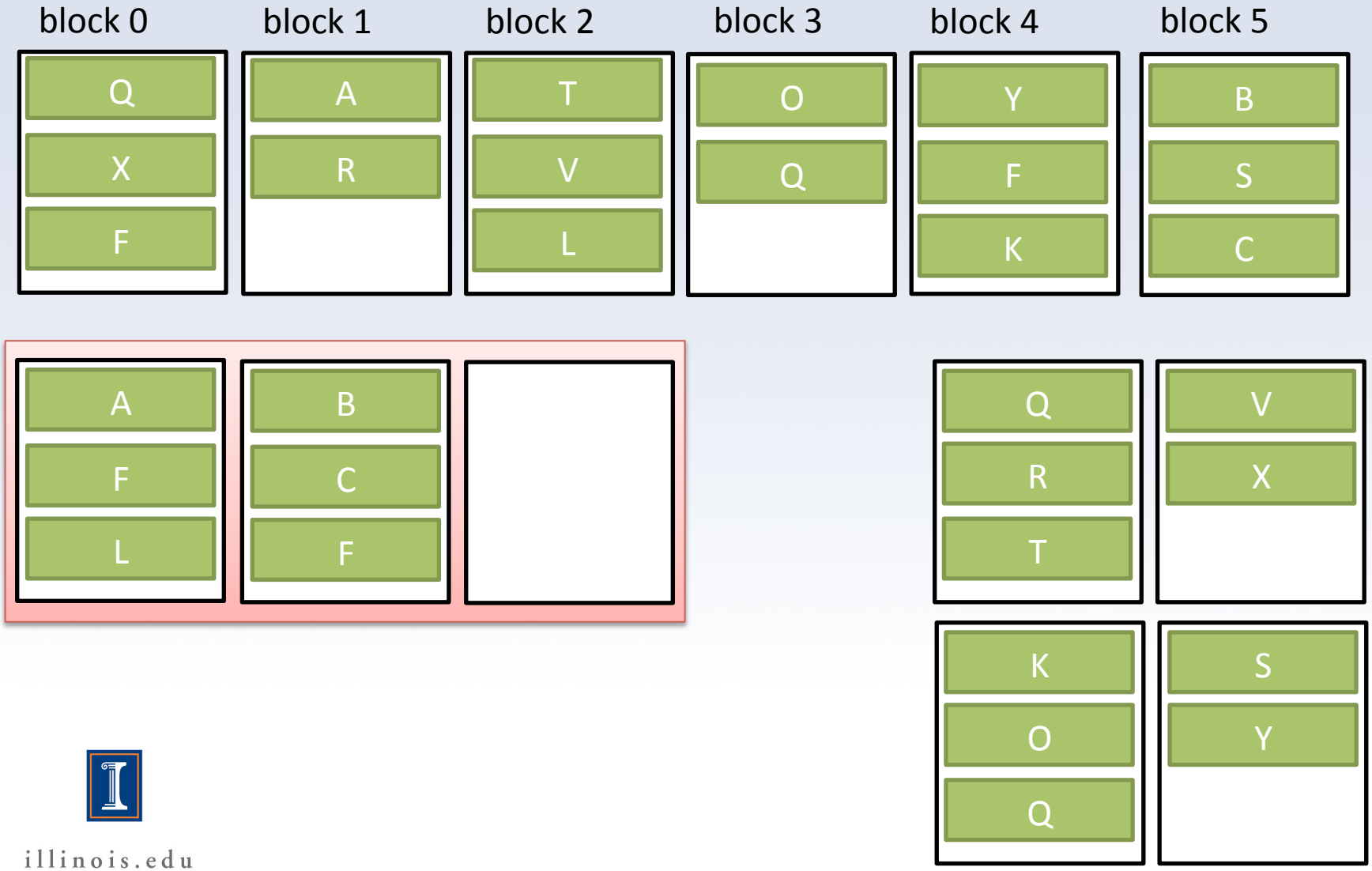
Phase 1


block 0
Q
X
F
block 1
A
R
block 2
T
V
L
block 3
O
Q
block 4
Y
F
K
block 5
B
S
C
A
F
L
Q
R
T
V
X
B
C
F
K
O
Q
S
Y

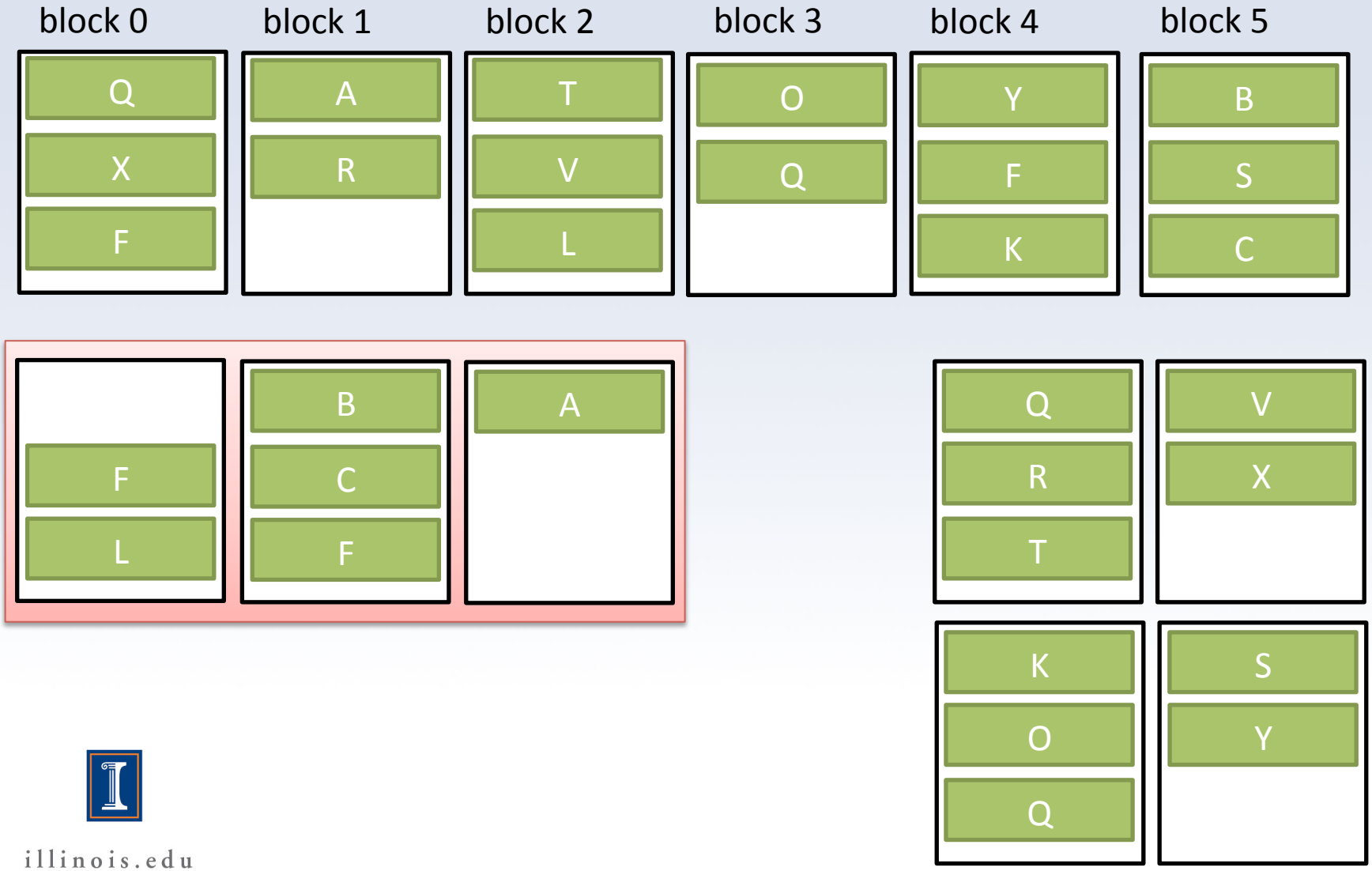
Phase 2


block 0
Q
X
F
block 1
A
R
block 2
T
V
L
block 3
O
Q
block 4
Y
F
K
block 5
B
S
C
sublist 1
sublist 2
output
A
F
L
Q
R
T
V
X
B
C
F
K
O
Q
S
Y

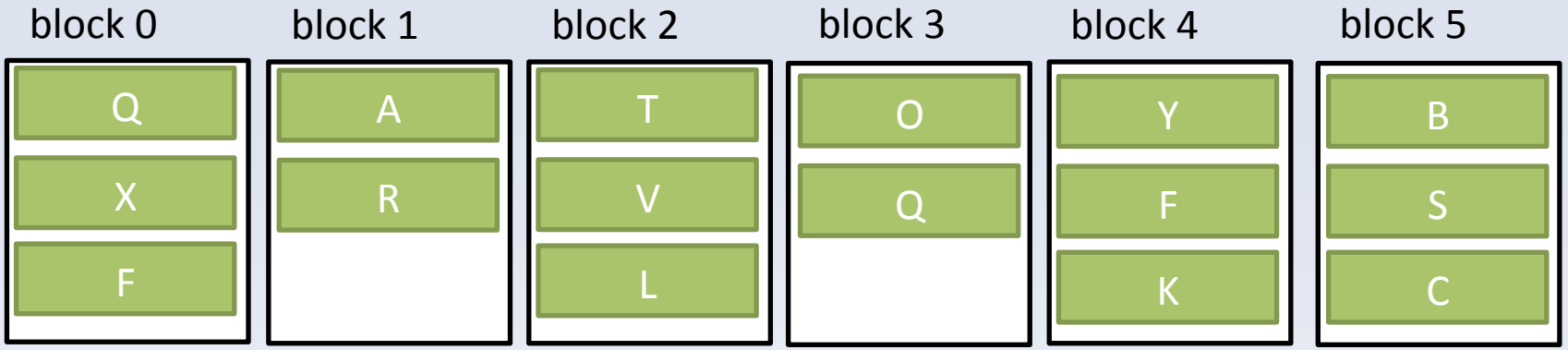
Phase 2

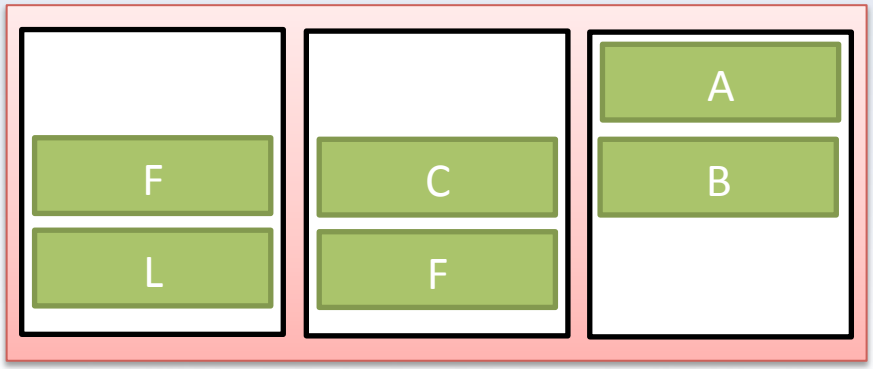

block 0
block 1
block 2
block 3
block 4
block 5
Q
X
F
A
R
T
V
L
O
Q
Y
F
K
B
S
C
A
F
L
B
C
F
Q
R
T
V
X
K
O
Q
S
Y

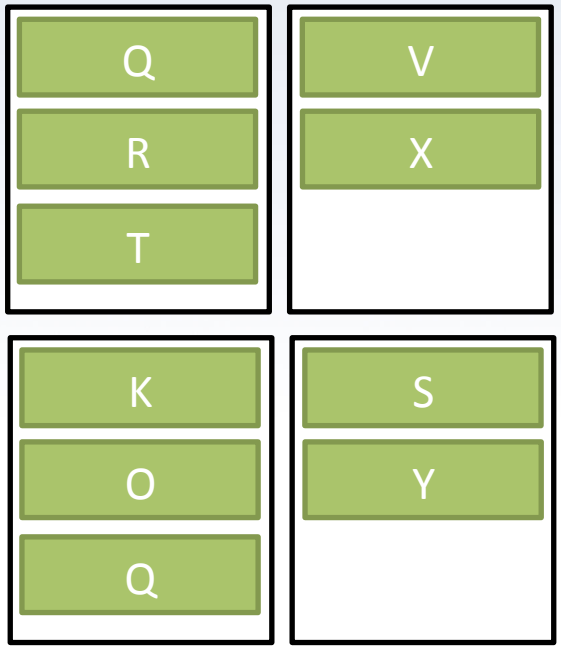
Phase 2


block 0
block 1
block 2
block 3
block 4
block 5
Q
X
F
A
R
T
V
L
O
Q
Y
F
K
B
S
C
F
L
B
C
F
A
Q
R
T
V
X
K
O
Q
S
Y

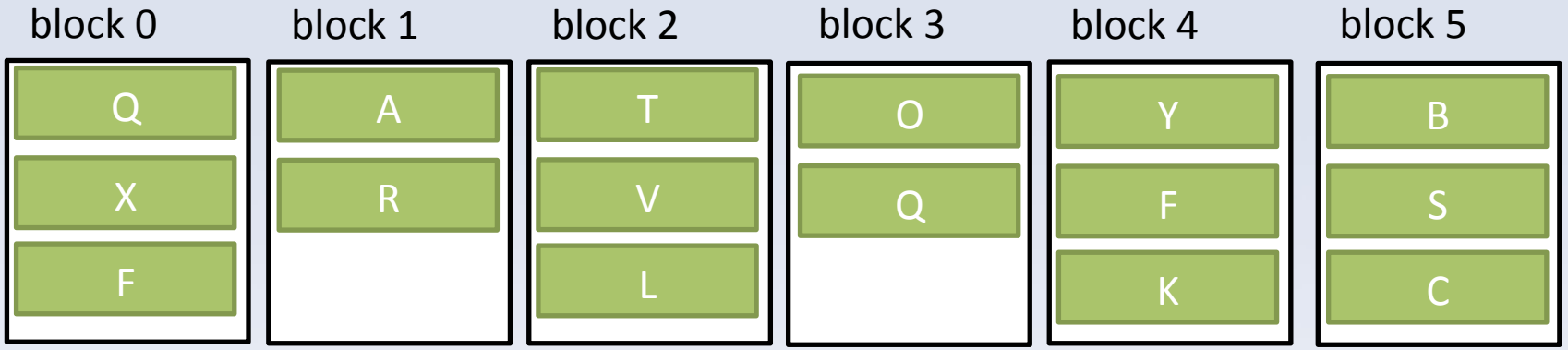
Phase 2

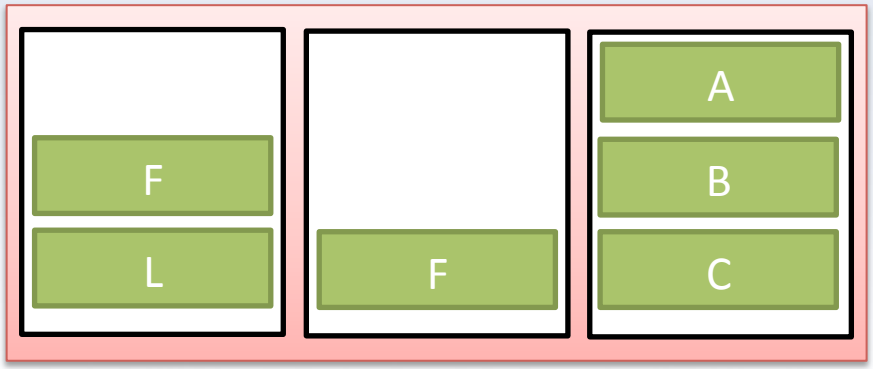

block 0
block 1
block 2
block 3
block 4
block 5
Q
X
F
A
R
T
V
L
O
Q
Y
F
K
B
S
C

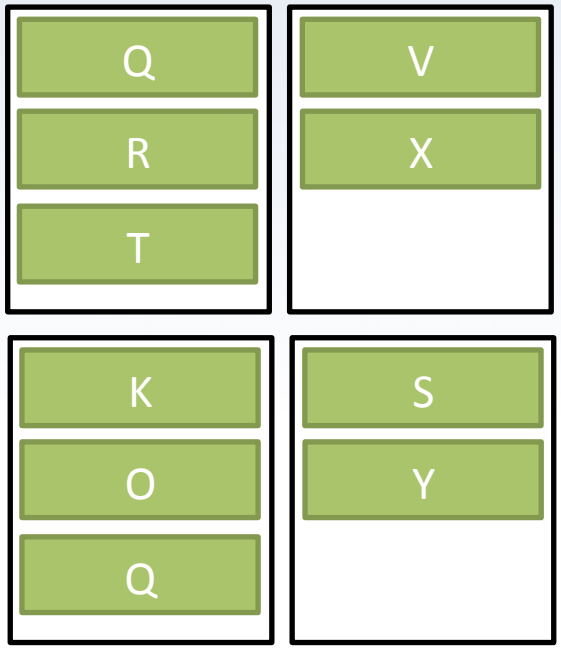

F
L
C
F
A
B


Q
R
T
V
X
K
O
Q
S
Y

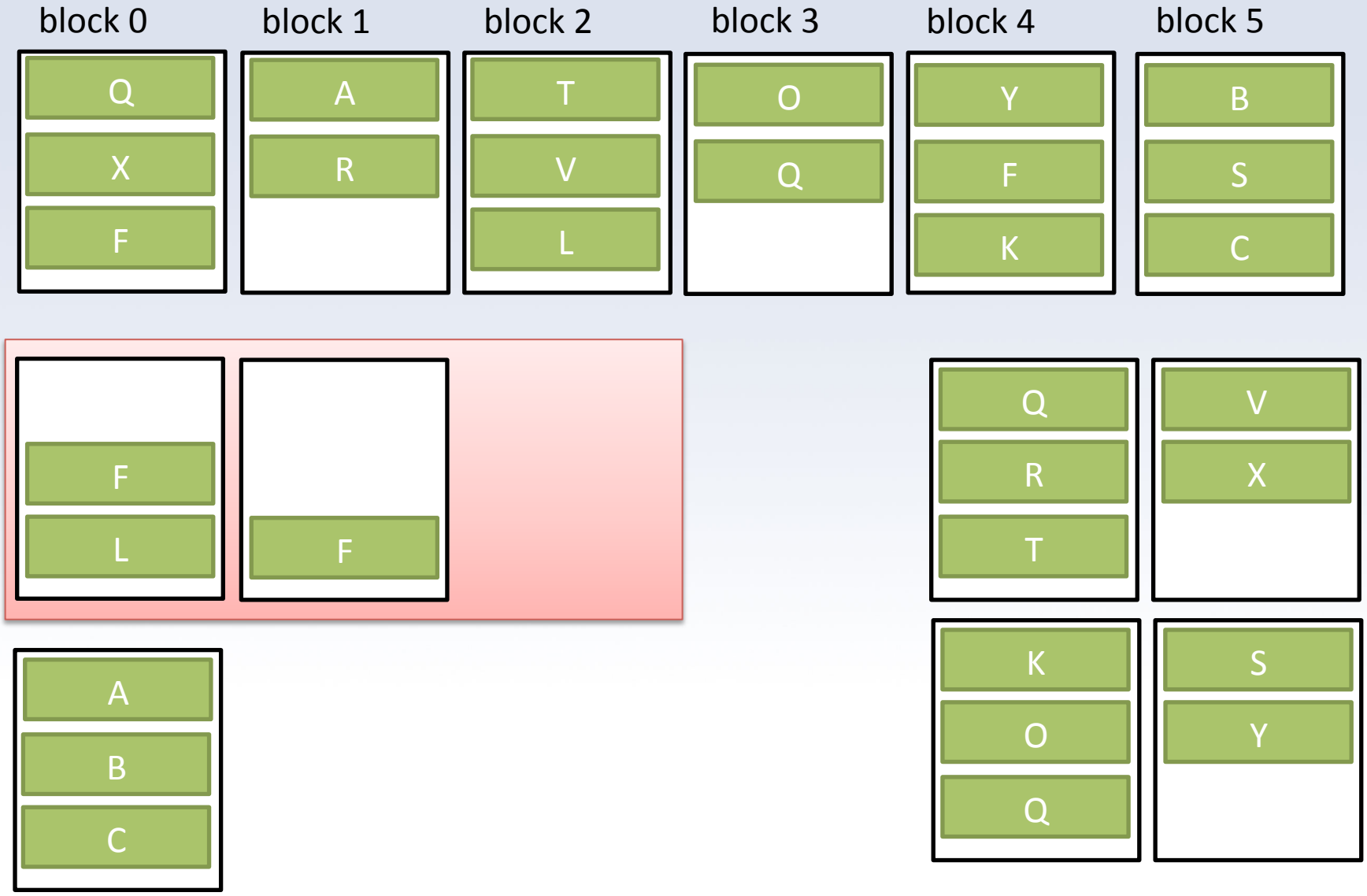
Phase 2


block 0
Q
X
F
block 1
A
R
block 2
T
V
L
block 3
O
Q
block 4
Y
F
K
block 5
B
S
C

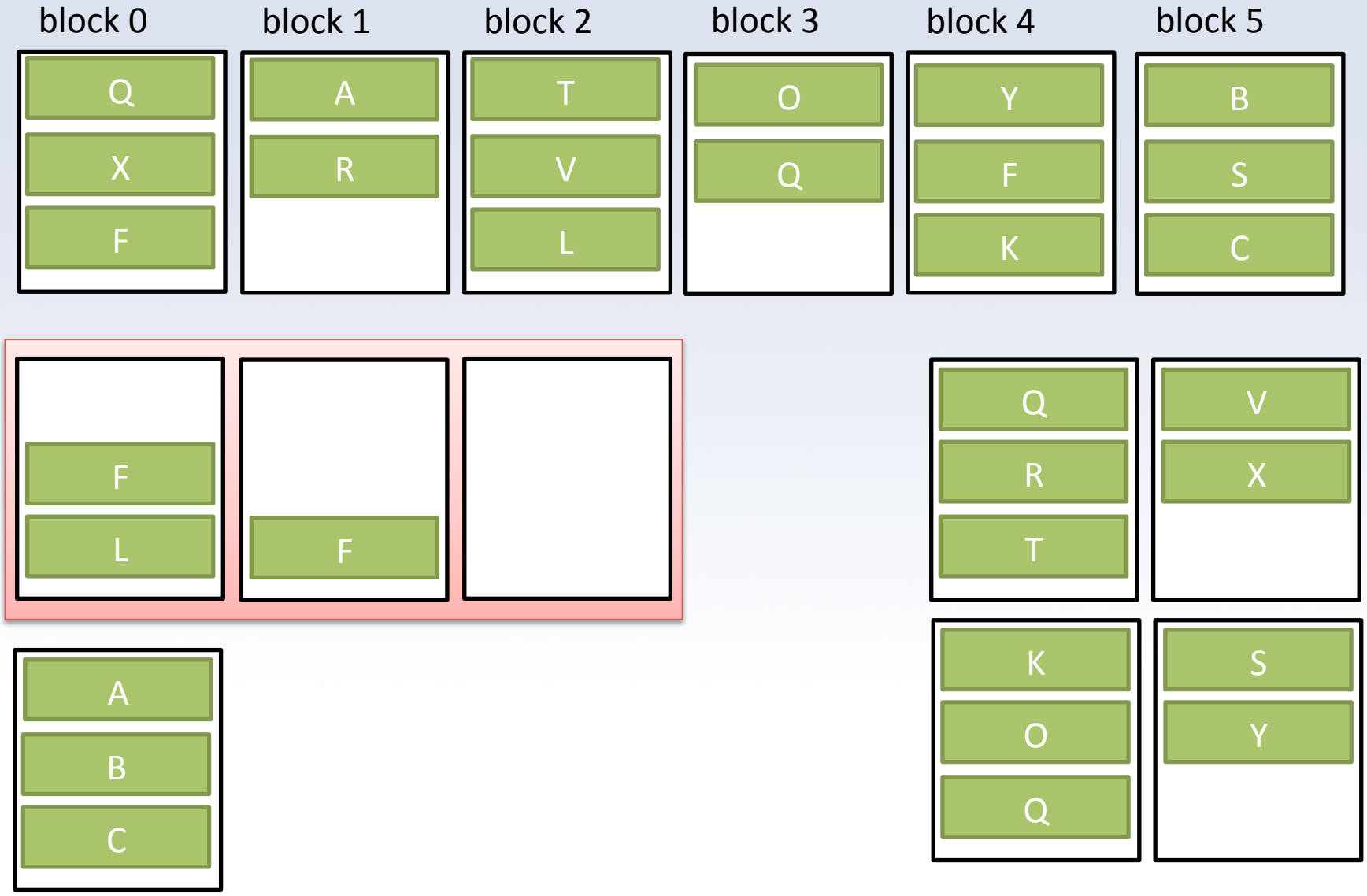

F
L
F
A
B
C


Q
R
T
V
X
K
O
Q
S
Y

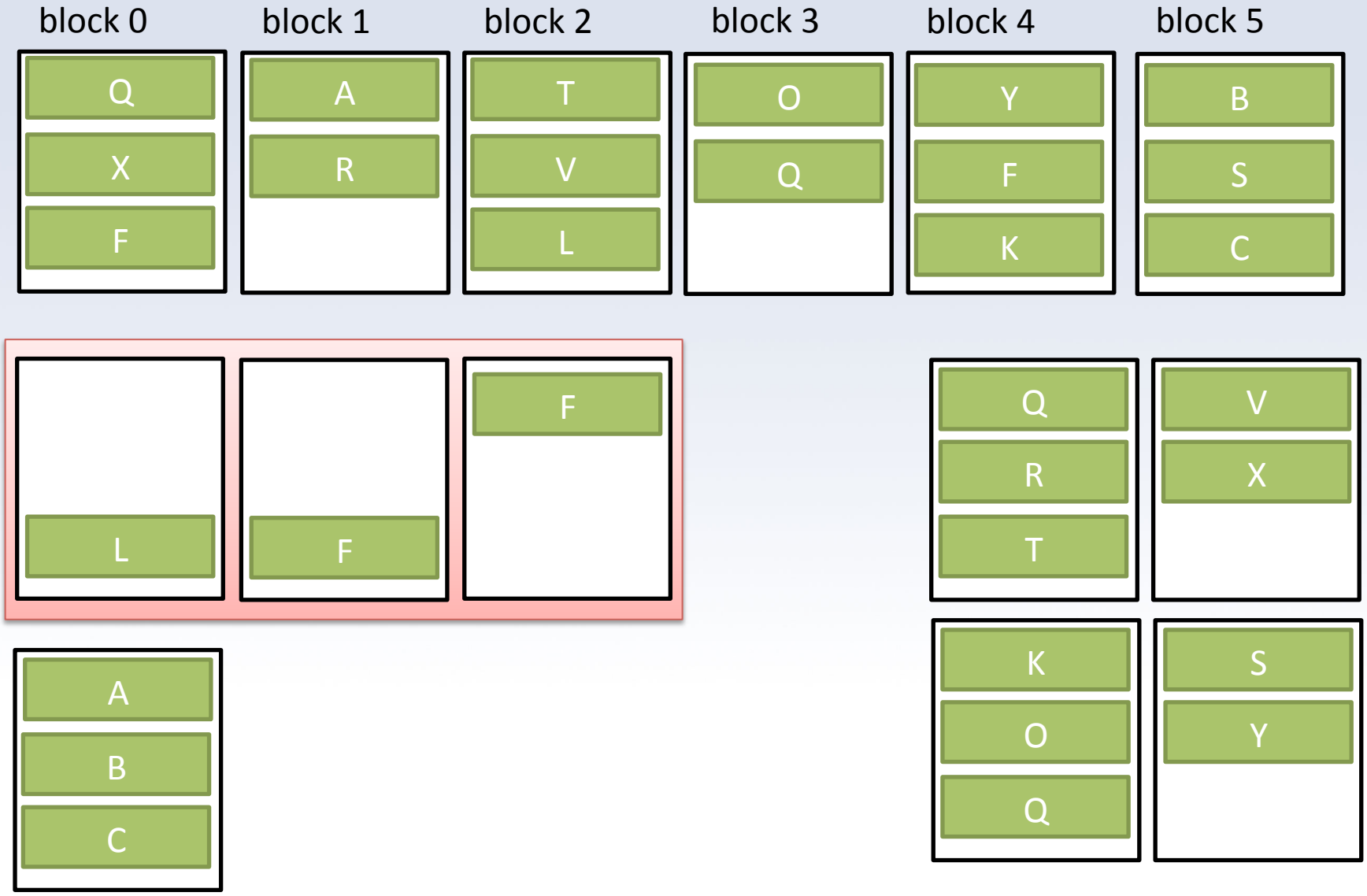
Phase 2


block 0
Q
X
F
block 1
A
R
block 2
T
V
L
block 3
O
Q
block 4
Y
F
K
block 5
B
S
C
F
L
F
Q
R
T
V
X
A
B
C
K
O
Q
S
Y

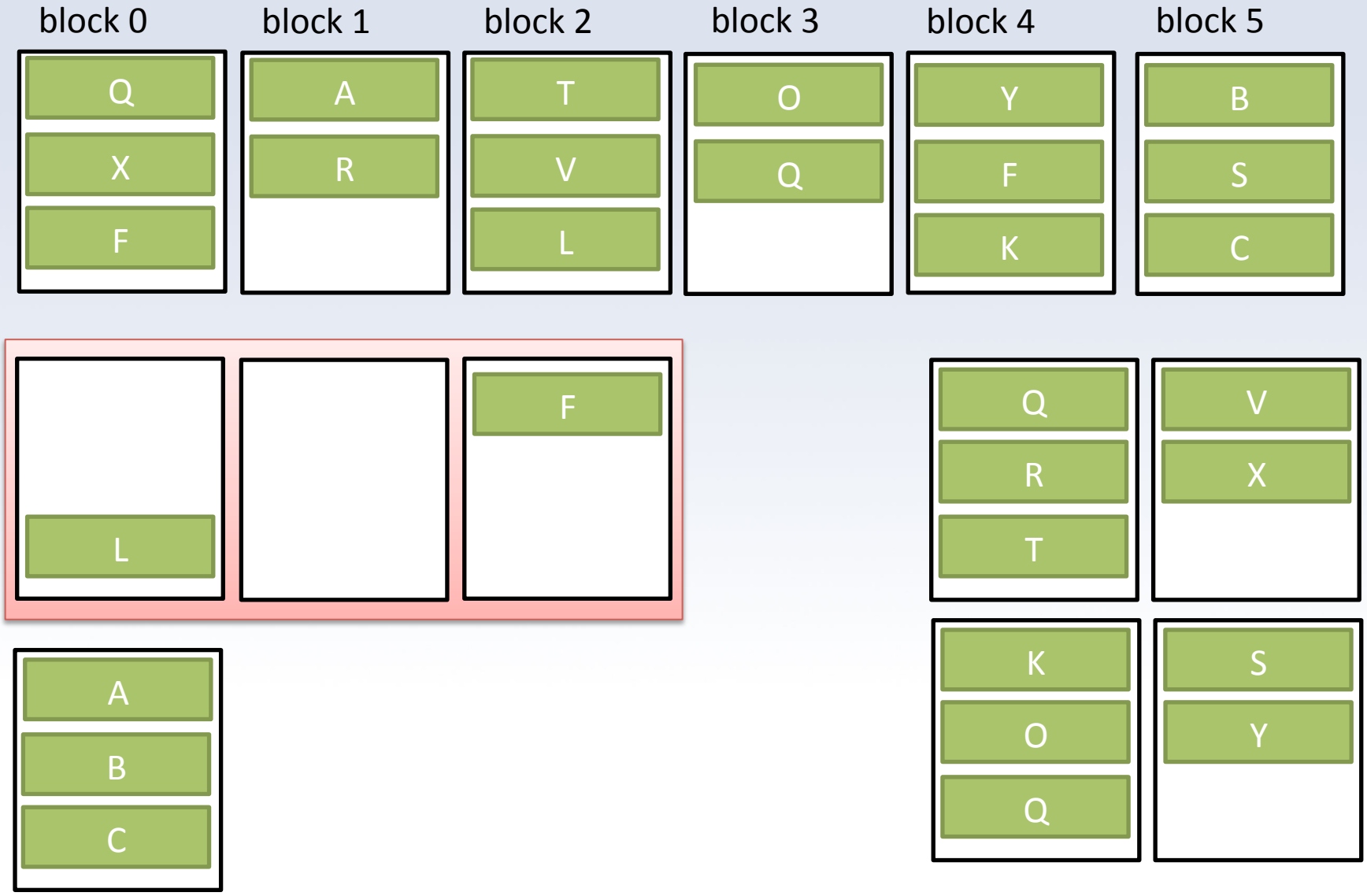
Phase 2


block 0
block 1
block 2
block 3
block 4
block 5
Q
X
F
A
R
T
V
L
O
Q
Y
F
K
B
S
C
F
L
F
Q
R
T
V
X
A
B
C
K
O
Q
S
Y

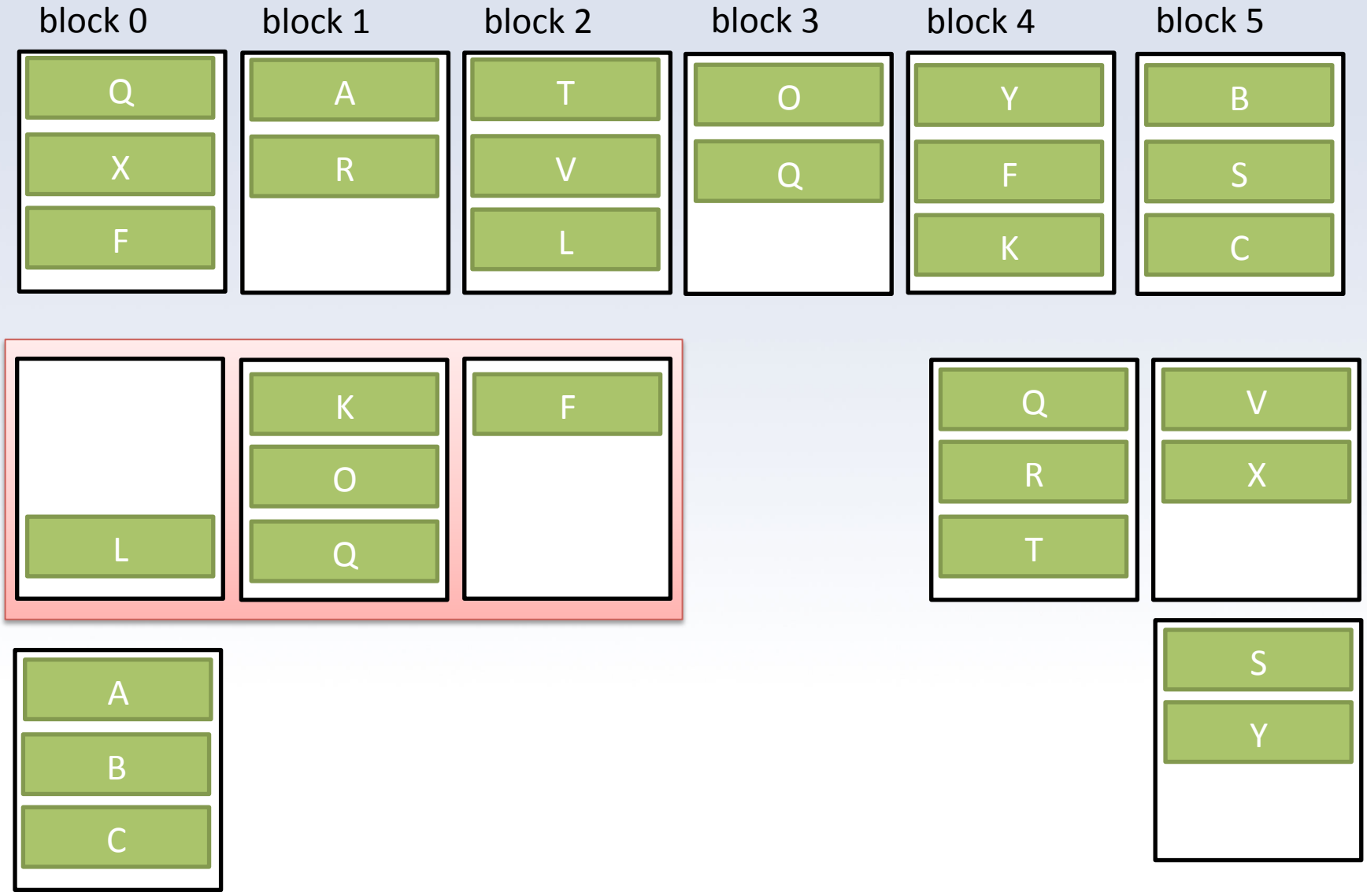
Phase 2


block 0
block 1
block 2
block 3
block 4
block 5
Q
X
F
A
R
T
V
L
O
Q
Y
F
K
B
S
C
L
F
F
Q
R
T
V
X
A
B
C
K
O
Q
S
Y

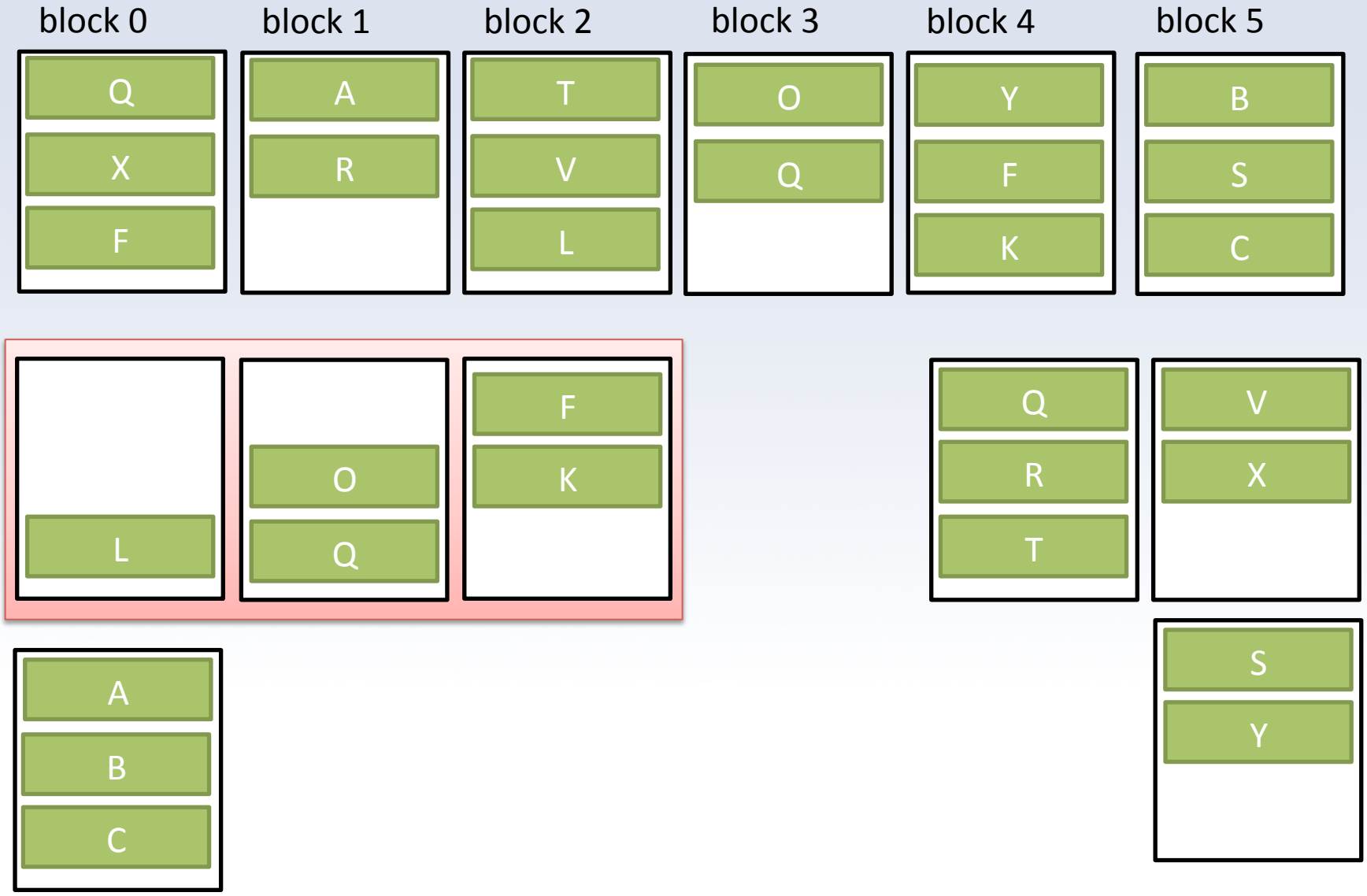
Phase 2


block 0
block 1
block 2
block 3
block 4
block 5
Q
X
F
A
R
T
V
L
O
Q
Y
F
K
B
S
C
L
F
Q
R
T
V
X
A
B
C
K
O
Q
S
Y

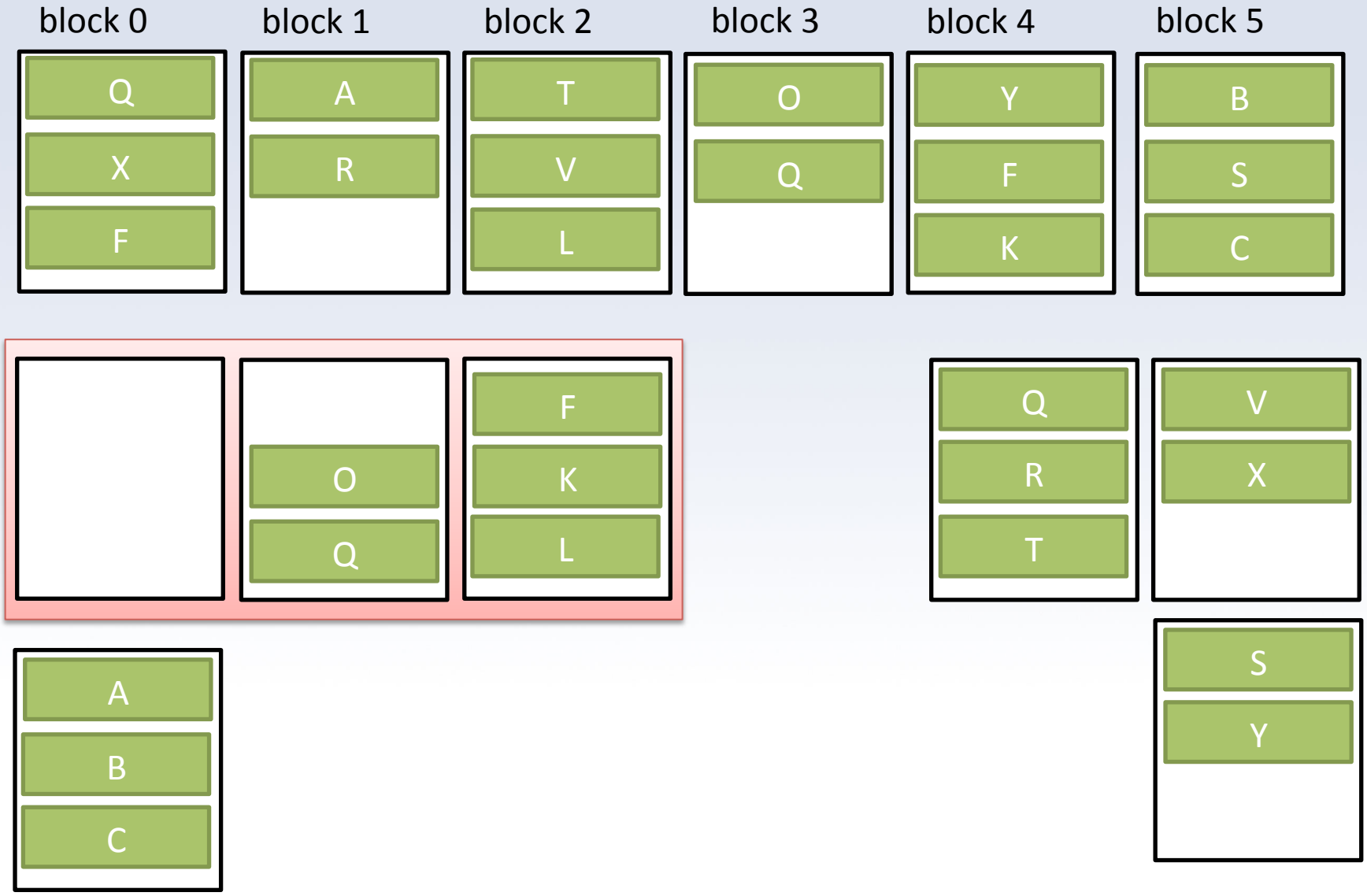
Phase 2


block 0
block 1
block 2
block 3
block 4
block 5
Q
X
F
A
R
T
V
L
O
Q
Y
F
K
B
S
C
L
K
O
Q
F
Q
R
T
V
X
A
B
C
S
Y

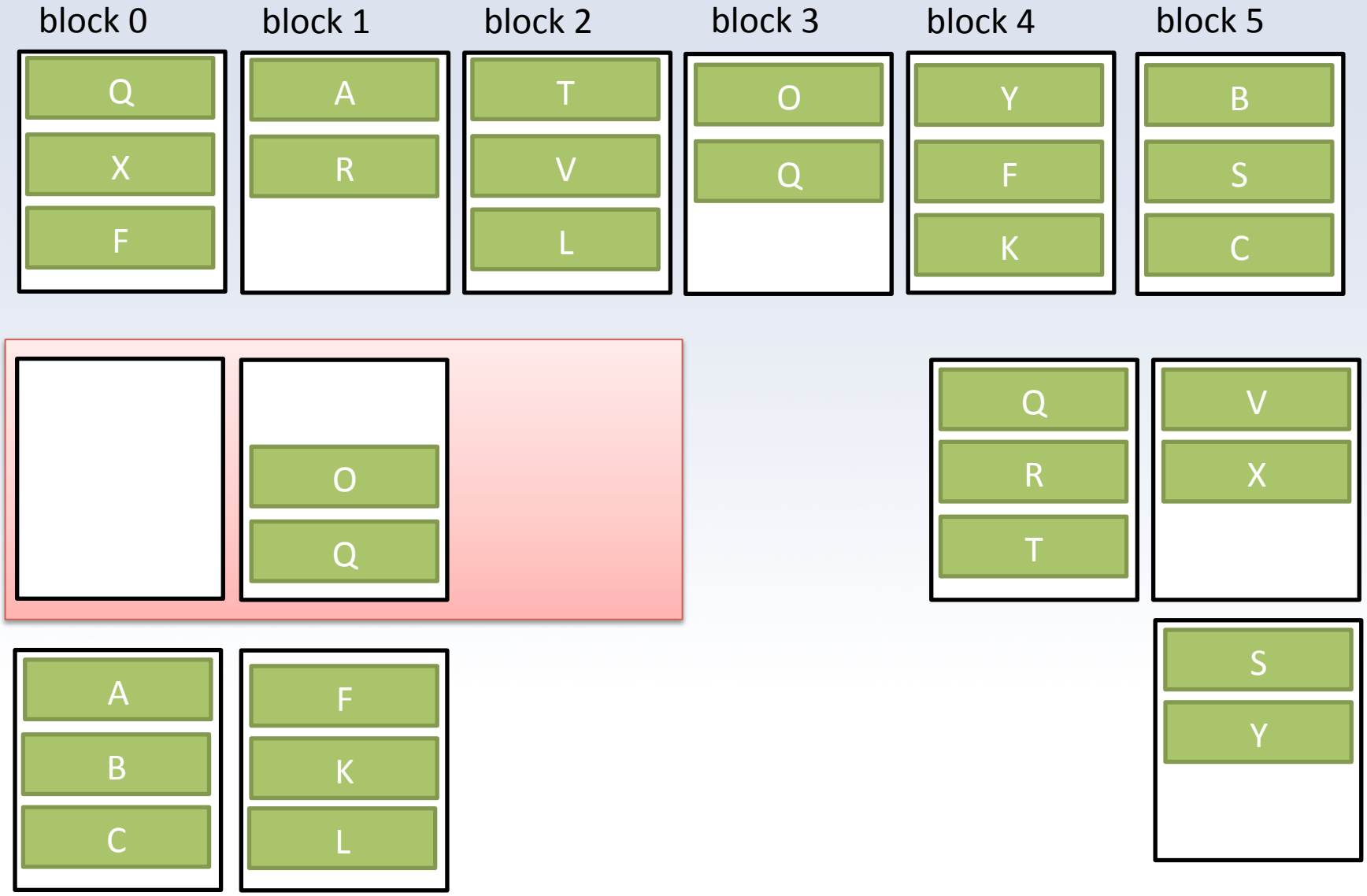
Phase 2


block 0
block 1
block 2
block 3
block 4
block 5
Q
X
F
A
R
T
V
L
O
Q
Y
F
K
B
S
C
L
O
Q
F
K
Q
R
T
V
X
A
B
C
S
Y

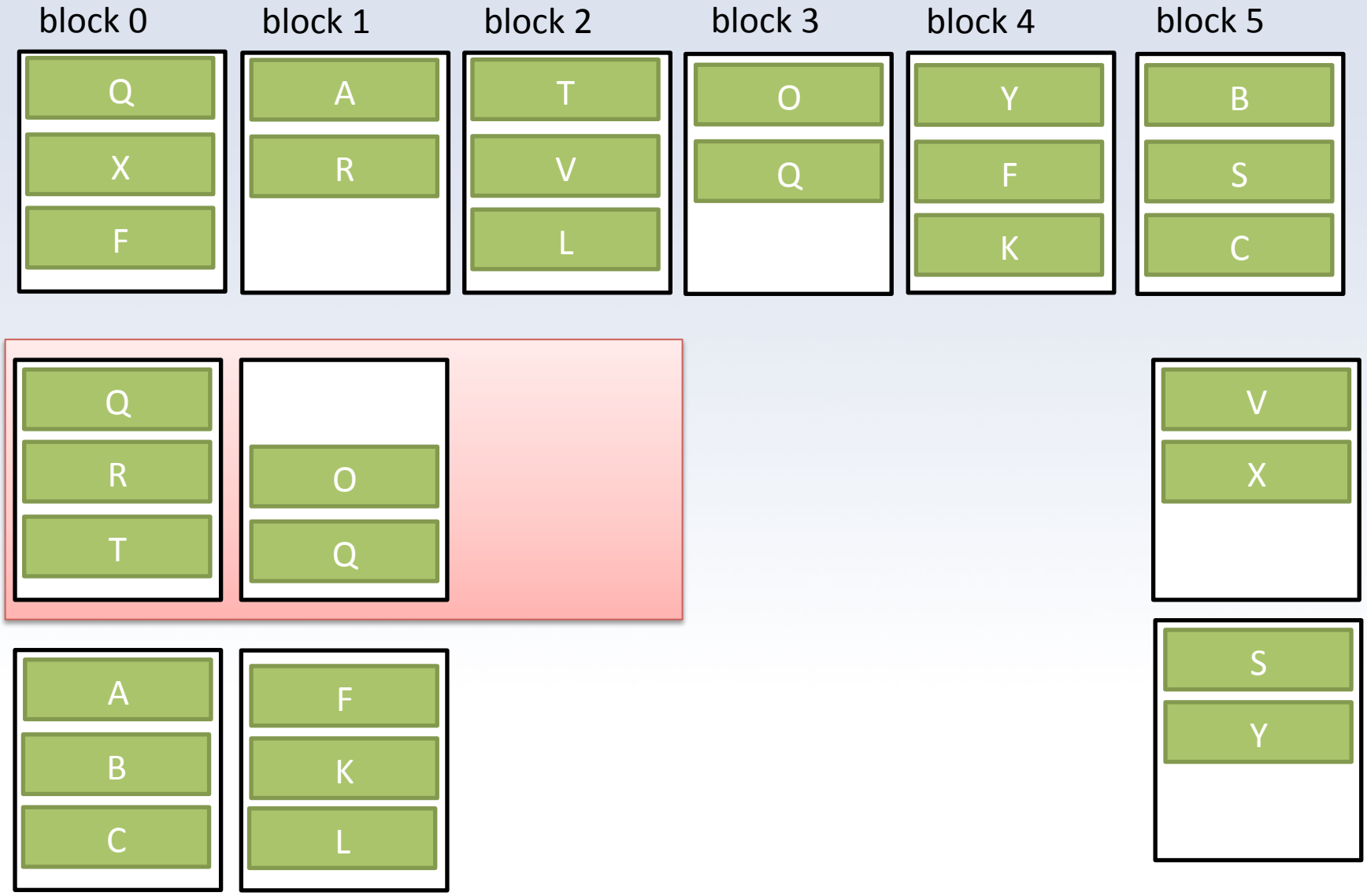
Phase 2


block 0
block 1
block 2
block 3
block 4
block 5
Q
X
F
A
R
T
V
L
O
Q
Y
F
K
B
S
C
O
Q
F
K
L
Q
R
T
V
X
A
B
C
S
Y

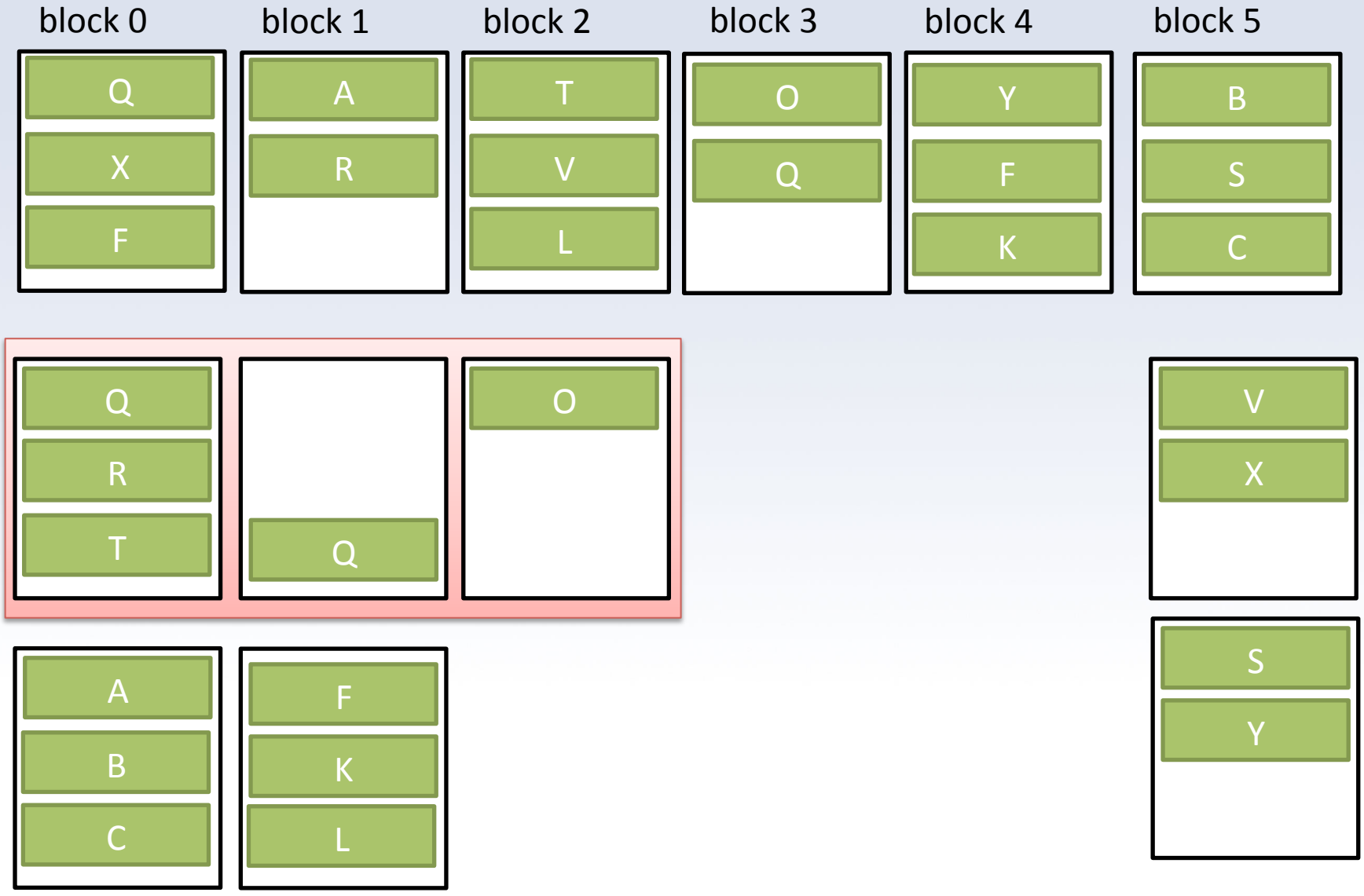
Phase 2


block 0
block 1
block 2
block 3
block 4
block 5
Q
X
F
A
R
T
V
L
O
Q
Y
F
K
B
S
C
O
Q
Q
R
T
V
X
A
B
C
F
K
L
S
Y

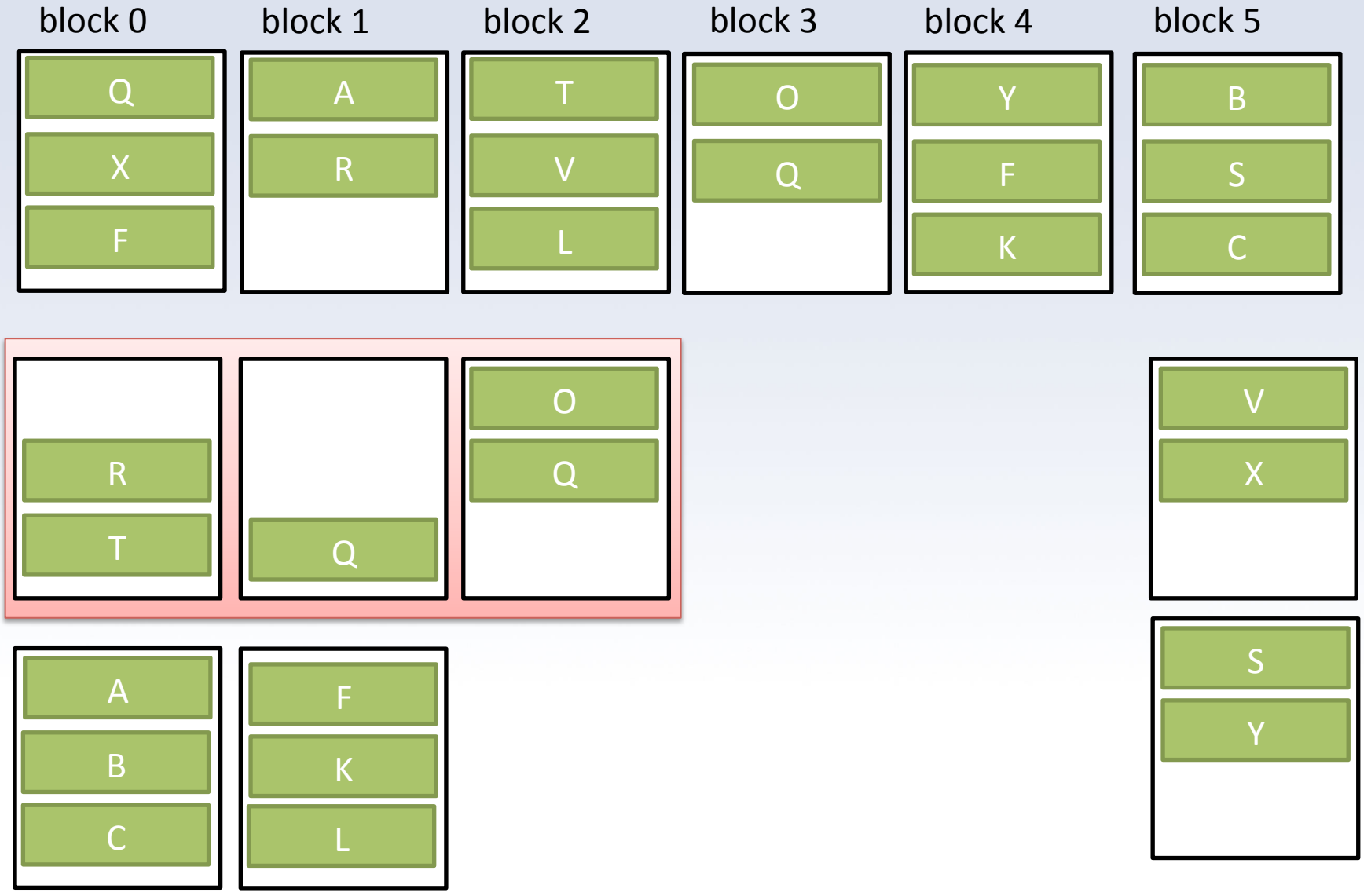
Phase 2


block 0
block 1
block 2
block 3
block 4
block 5
Q
X
F
A
R
T
V
L
O
Q
Y
F
K
B
S
C
Q
R
T
O
Q
V
X
A
B
C
F
K
L
S
Y

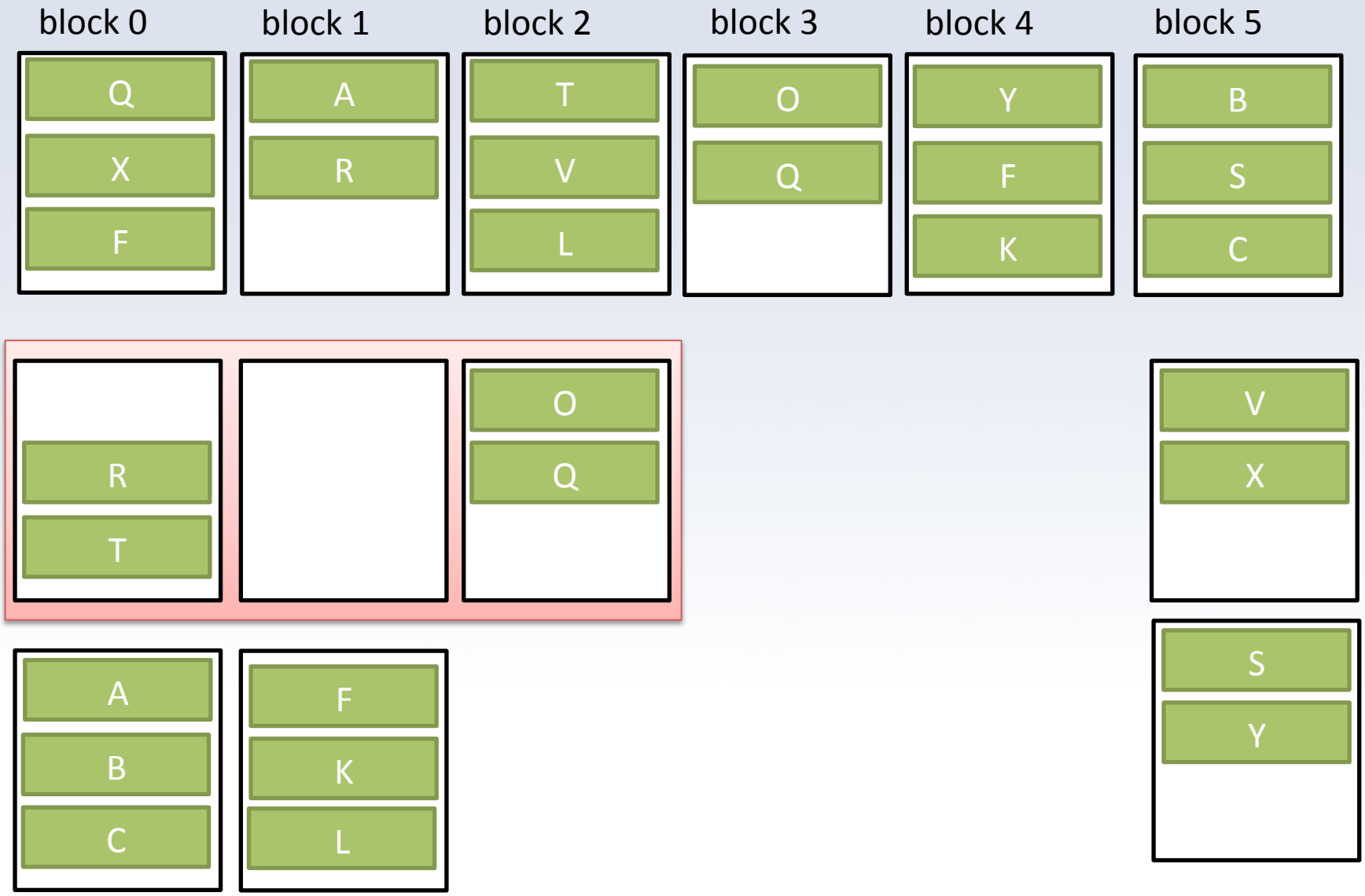
Phase 2


block 0
block 1
block 2
block 3
block 4
block 5
Q
X
F
A
R
T
V
L
O
Q
Y
F
K
B
S
C
Q
R
T
Q
O
V
X
A
B
C
F
K
L
S
Y

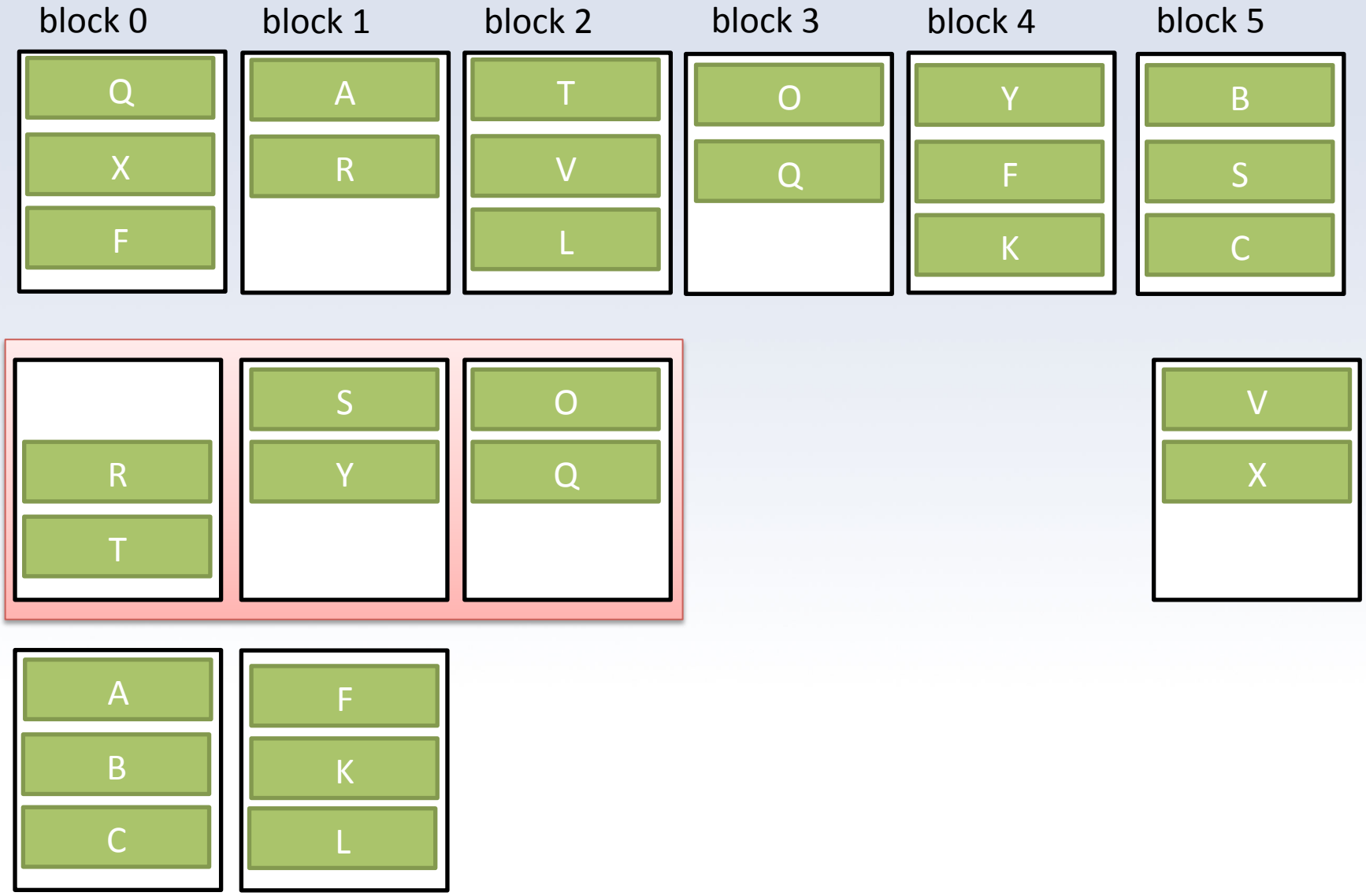
Phase 2


block 0
block 1
block 2
block 3
block 4
block 5
Q
X
F
A
R
T
V
L
O
Q
Y
F
K
B
S
C
R
T
Q
O
Q
V
X
A
B
C
F
K
L
S
Y

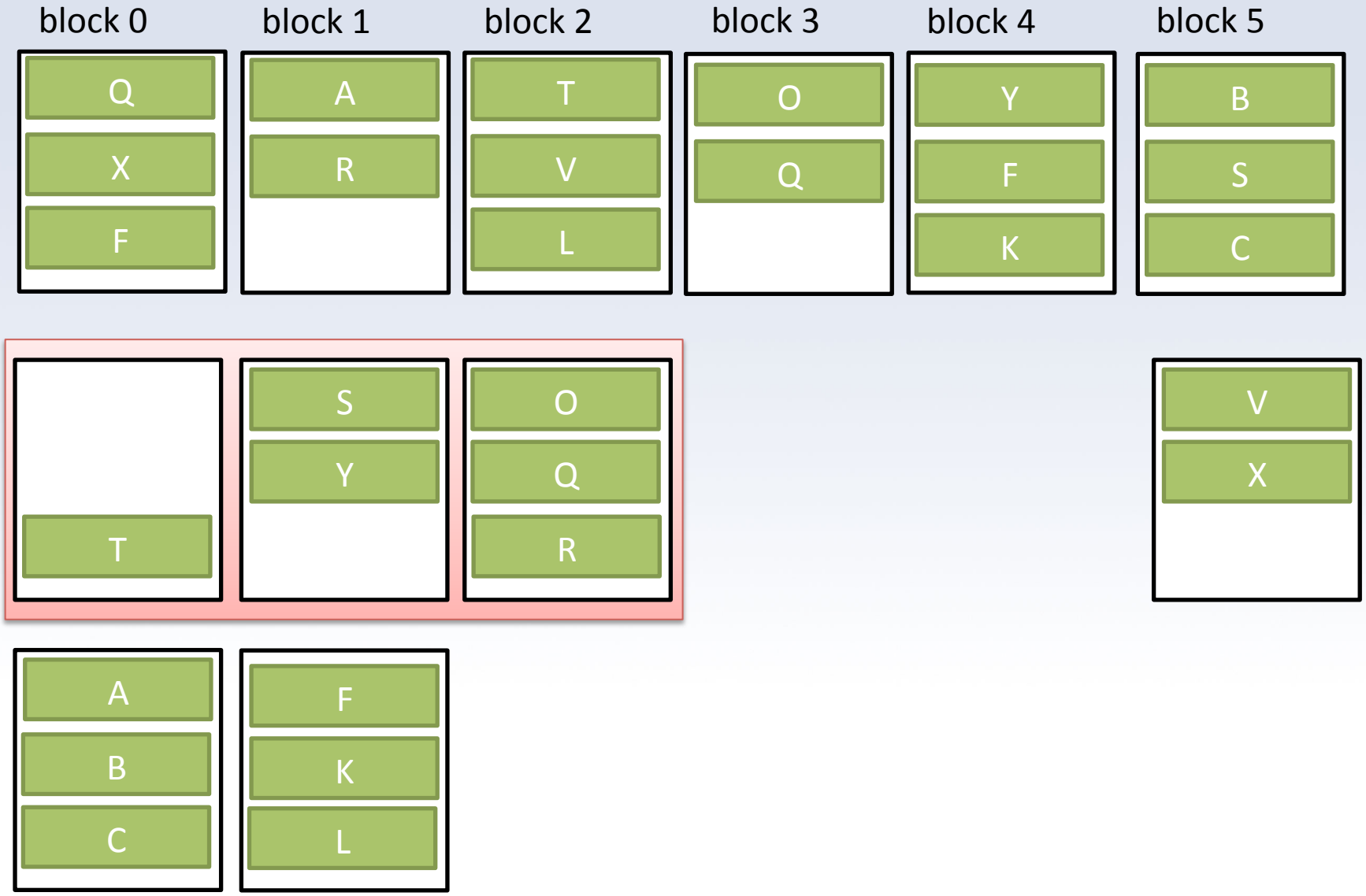
Phase 2


block 0
block 1
block 2
block 3
block 4
block 5
Q
X
F
A
R
T
V
L
O
Q
Y
F
K
B
S
C
R
T
O
Q
V
X
A
B
C
F
K
L
S
Y

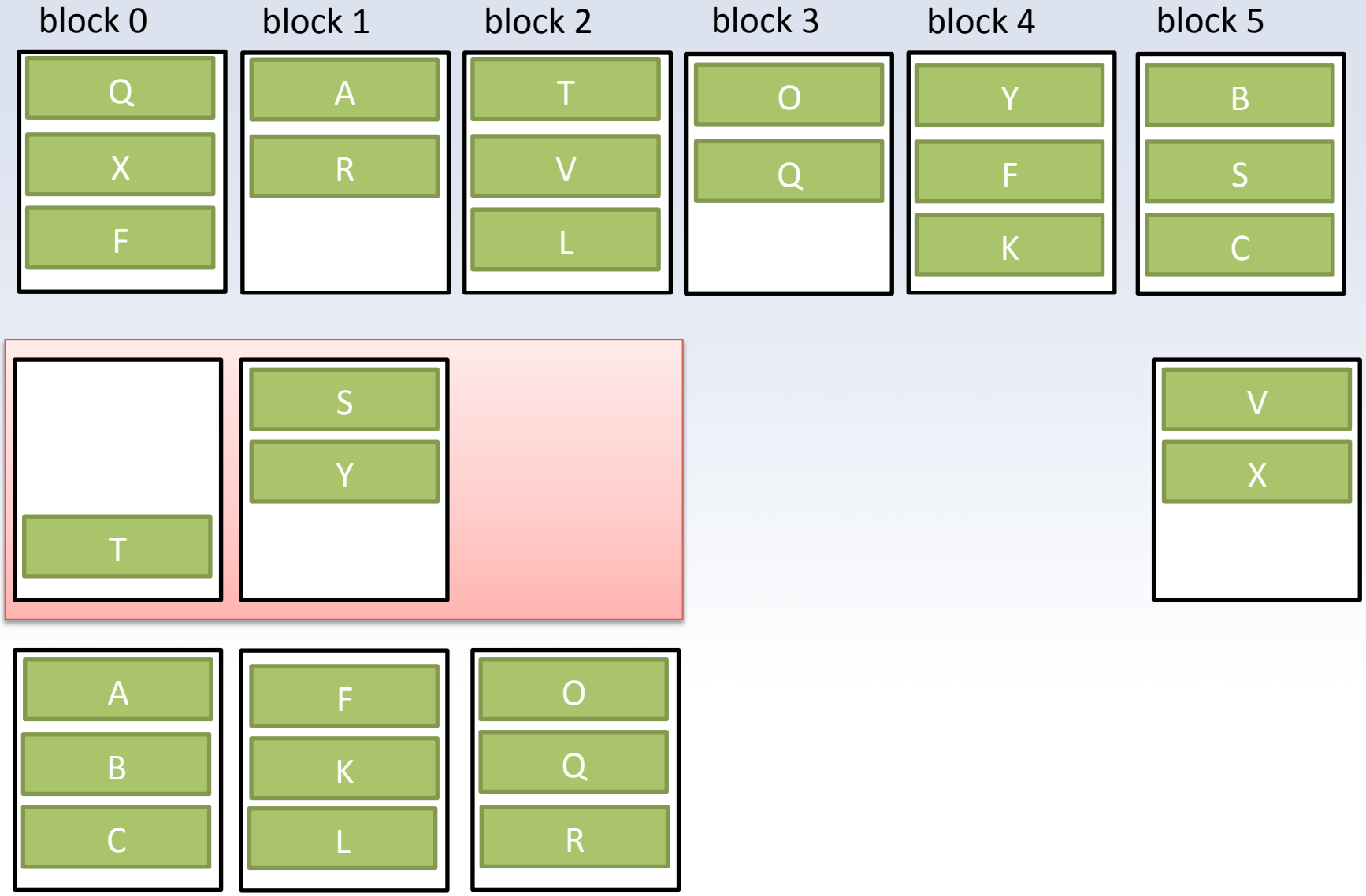
Phase 2


block 0
block 1
block 2
block 3
block 4
block 5
Q
X
F
A
R
T
V
L
O
Q
Y
F
K
B
S
C
R
T
S
Y
O
Q
V
X
A
B
C
F
K
L

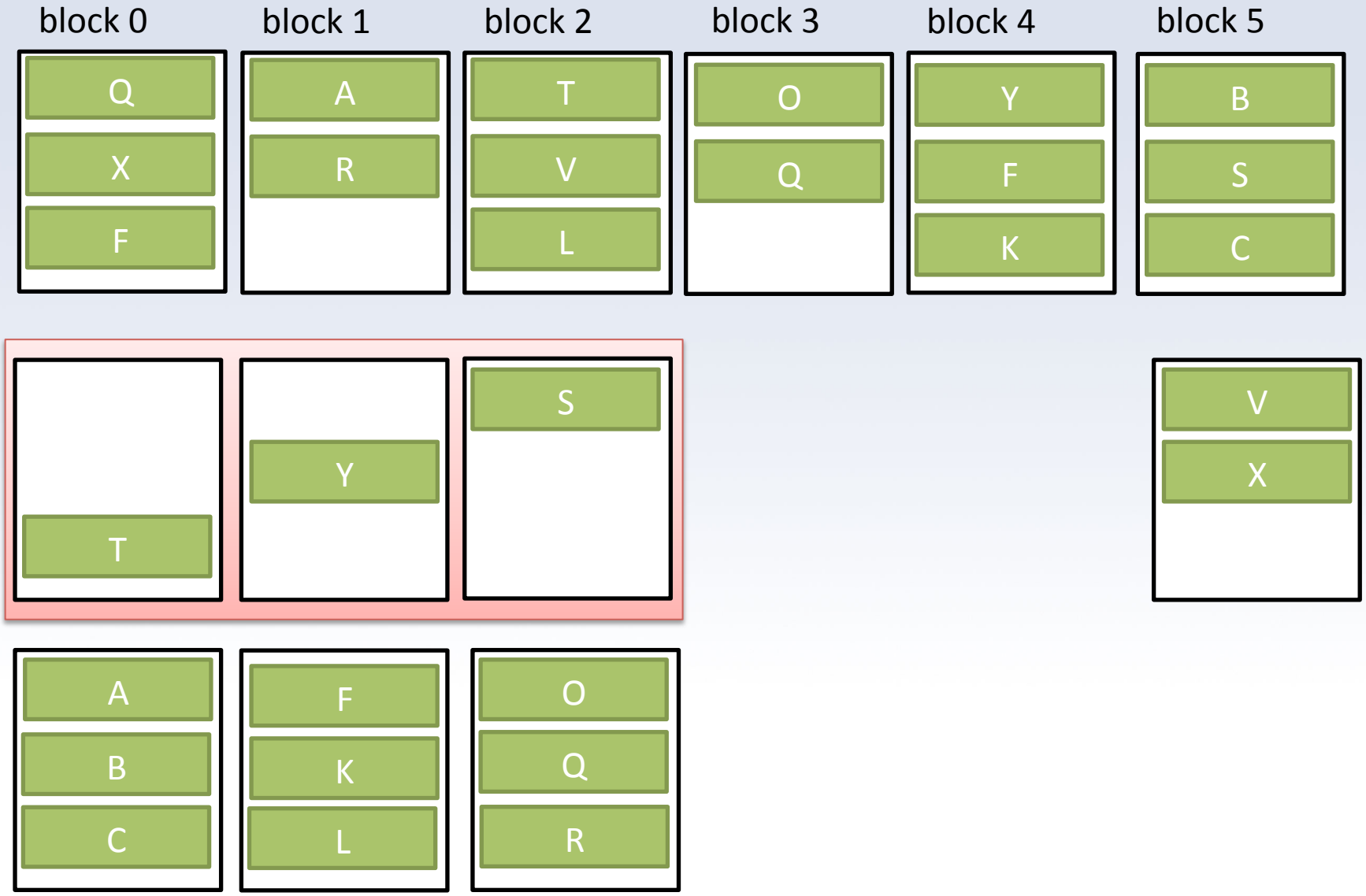
Phase 2


block 0
block 1
block 2
block 3
block 4
block 5
Q
X
F
A
R
T
V
L
O
Q
Y
F
K
B
S
C
T
S
Y
O
Q
R
V
X
A
B
C
F
K
L

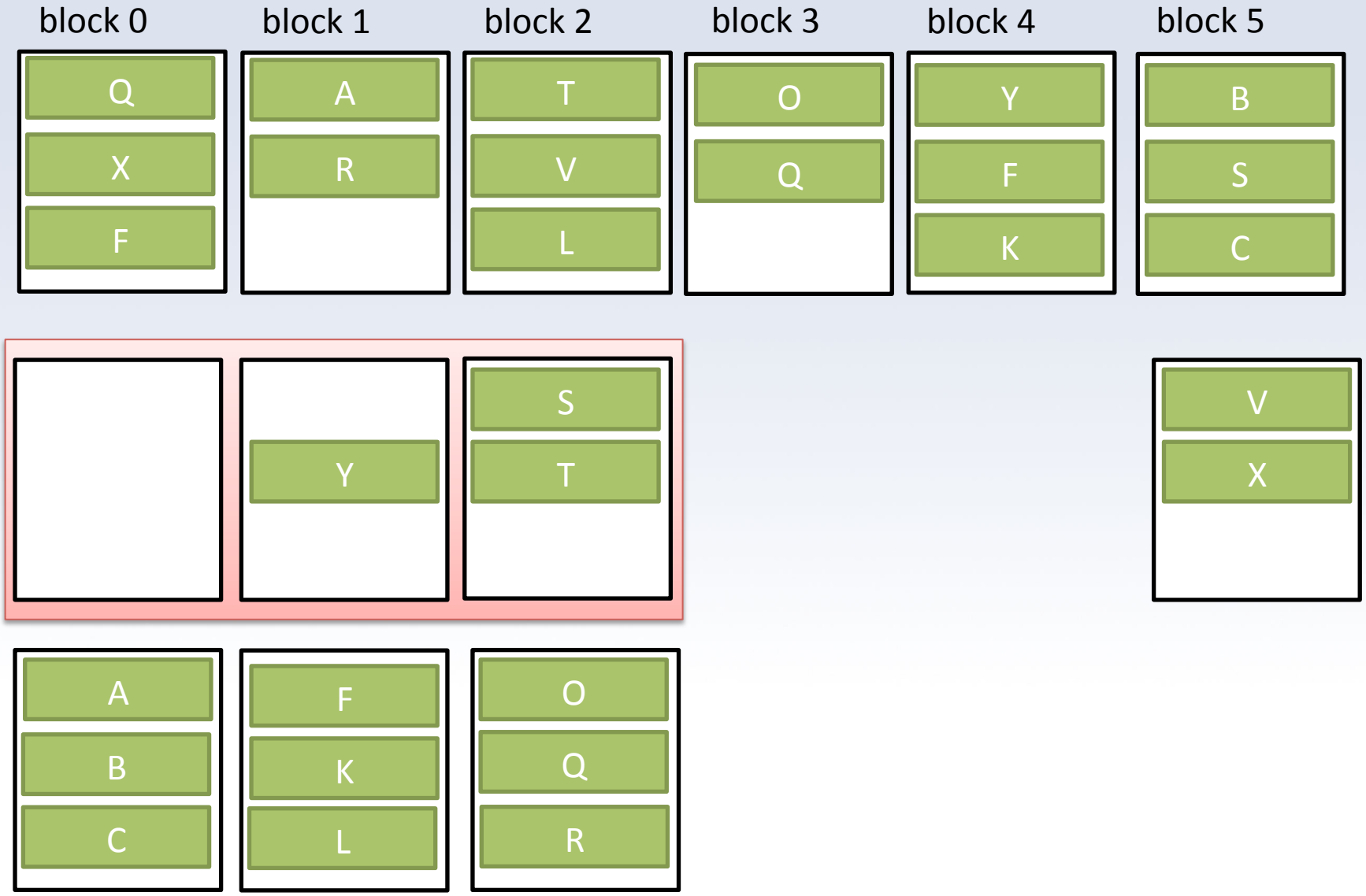
Phase 2


block 0
block 1
block 2
block 3
block 4
block 5
Q
X
F
A
R
T
V
L
O
Q
Y
F
K
B
S
C
T
S
Y
V
X
A
B
C
F
K
L
O
Q
R

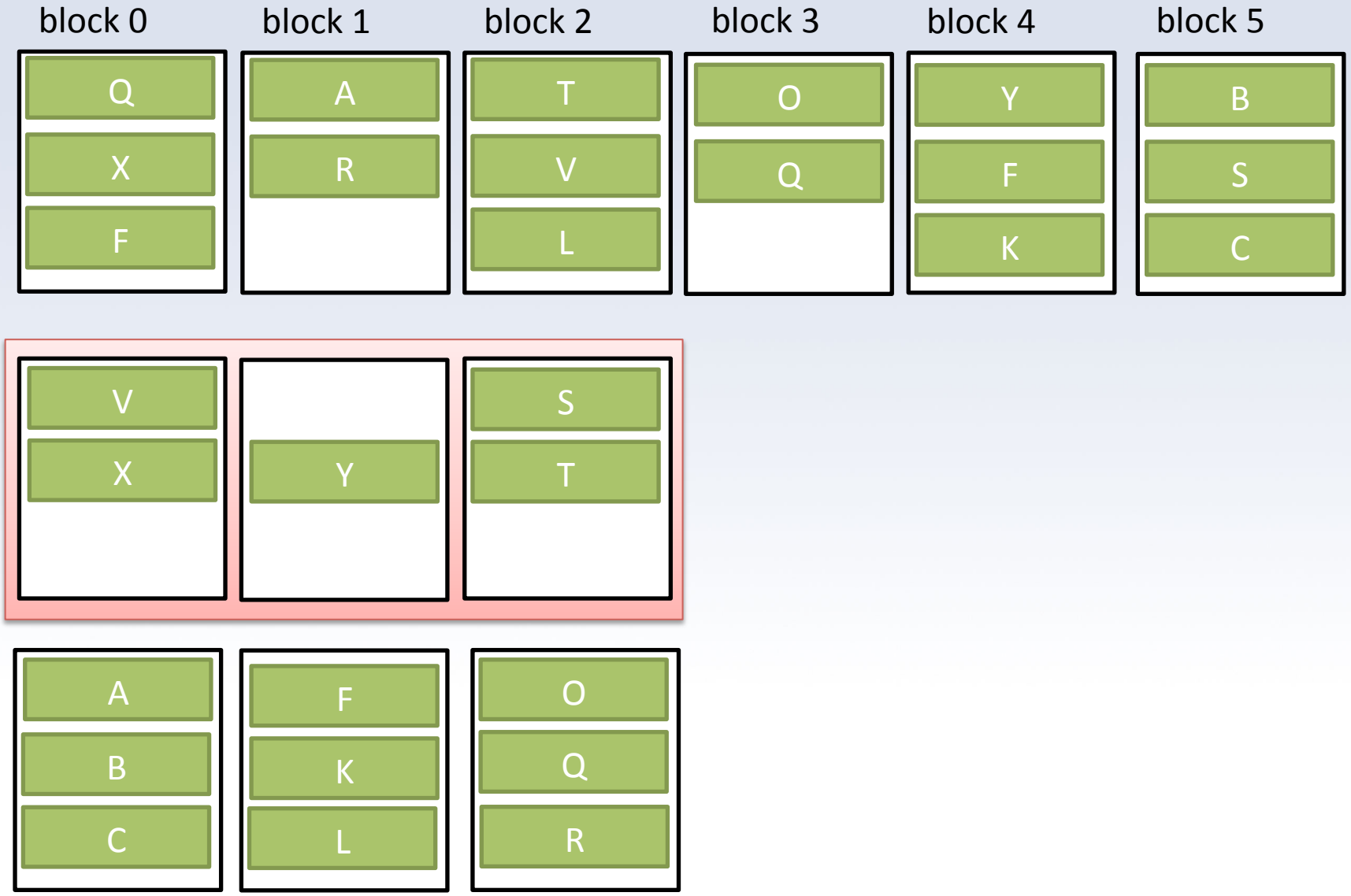
Phase 2


block 0
block 1
block 2
block 3
block 4
block 5
Q
X
F
A
R
T
V
L
O
Q
Y
F
K
B
S
C
T
Y
S
V
X
A
B
C
F
K
L
O
Q
R

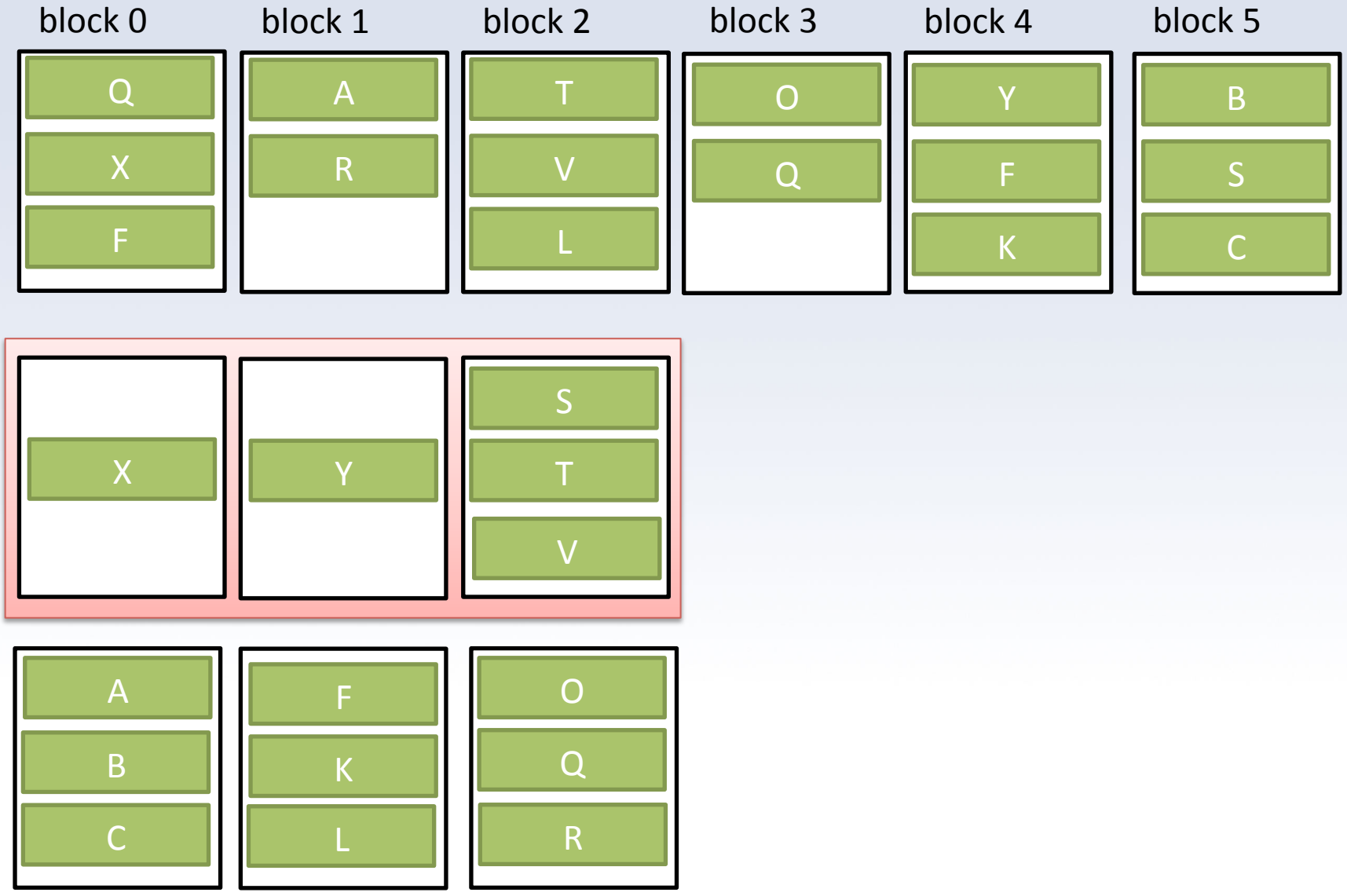
Phase 2


block 0
block 1
block 2
block 3
block 4
block 5
Q
X
F
A
R
T
V
L
O
Q
Y
F
K
B
S
C
Y
S
T
V
X
A
B
C
F
K
L
O
Q
R

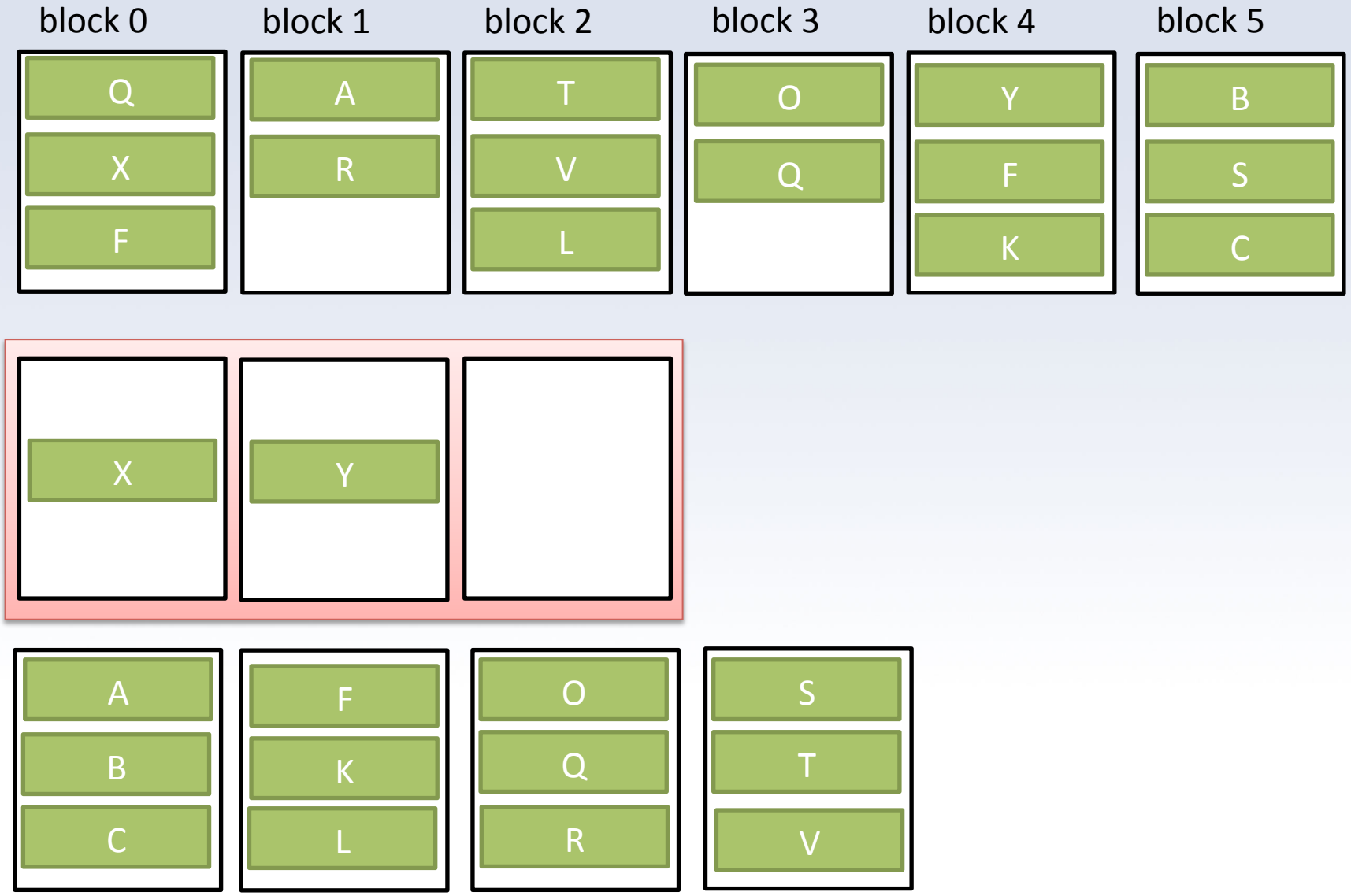
Phase 2


block 0
Q
X
F
block 1
A
R
block 2
T
V
L
block 3
O
Q
block 4
Y
F
K
block 5
B
S
C
V
X
Y
S
T
A
B
C
F
K
L
O
Q
R

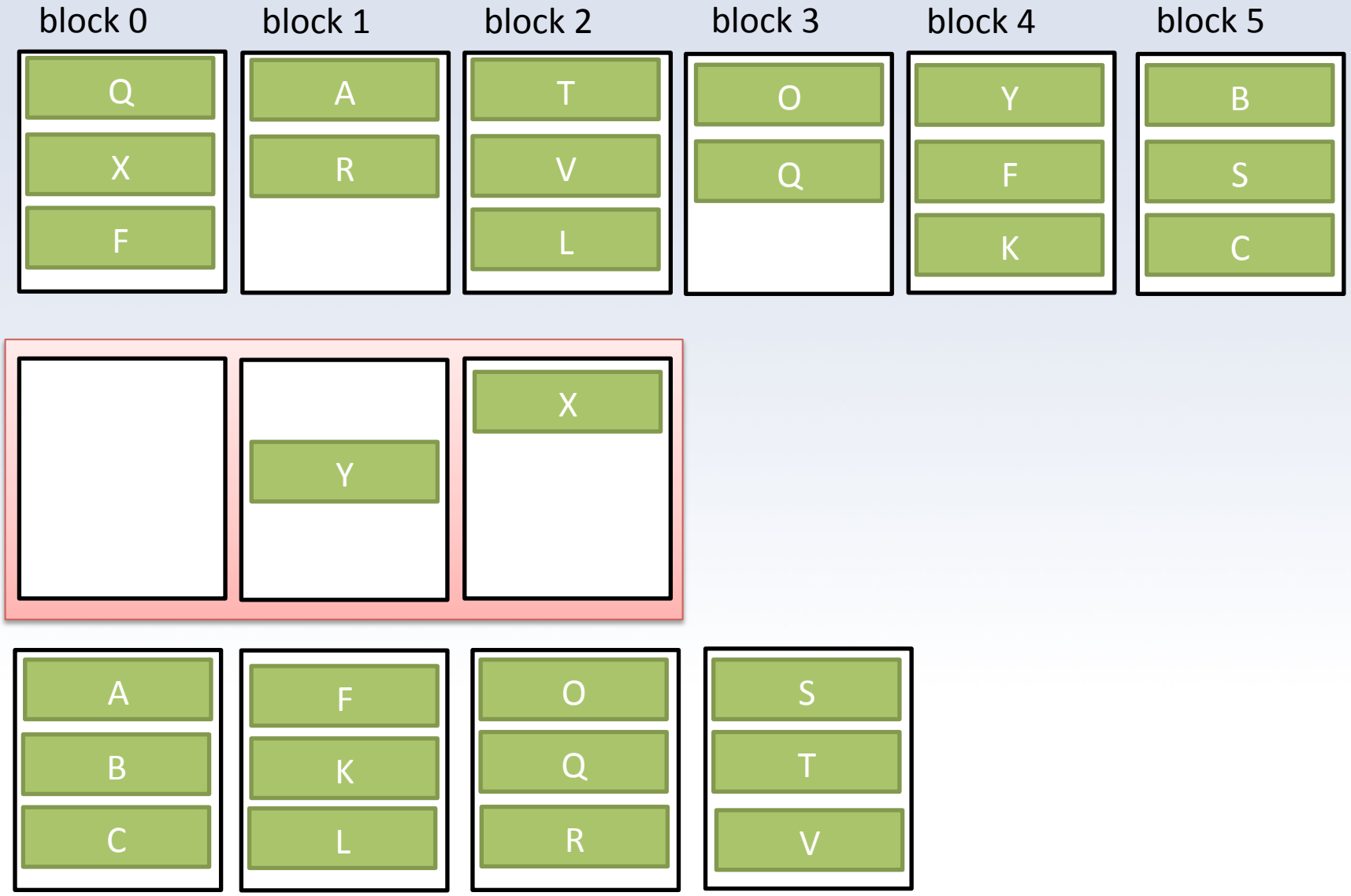
Phase 2


block 0
block 1
block 2
block 3
block 4
block 5
Q
X
F
A
R
T
V
L
O
Q
Y
F
K
B
S
C
X
Y
S
T
V
A
B
C
F
K
L
O
Q
R

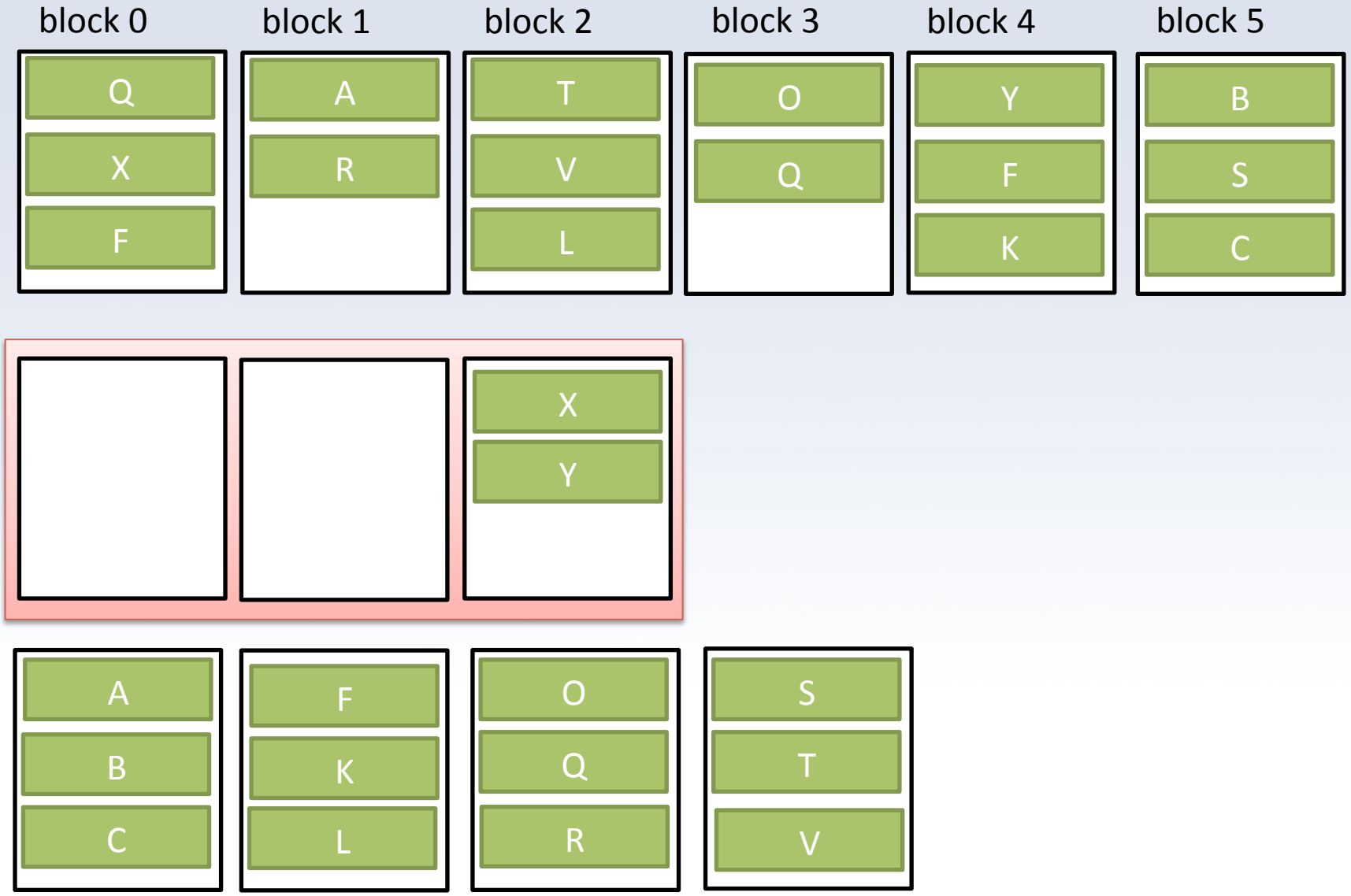
Phase 2


block 0
block 1
block 2
block 3
block 4
block 5
Q
X
F
A
R
T
V
L
O
Q
Y
F
K
B
S
C
X
Y
A
B
C
F
K
L
O
Q
R
S
T
V

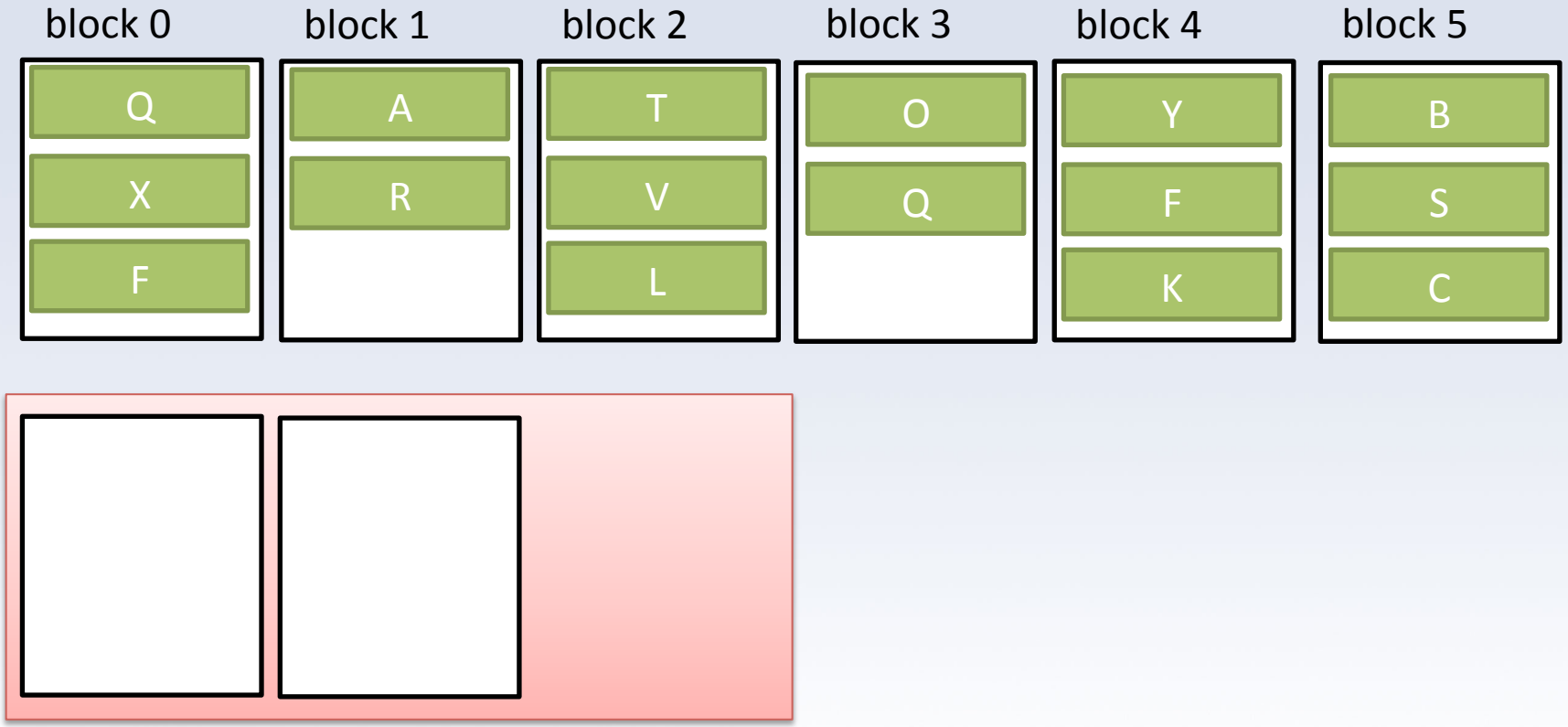
Phase 2

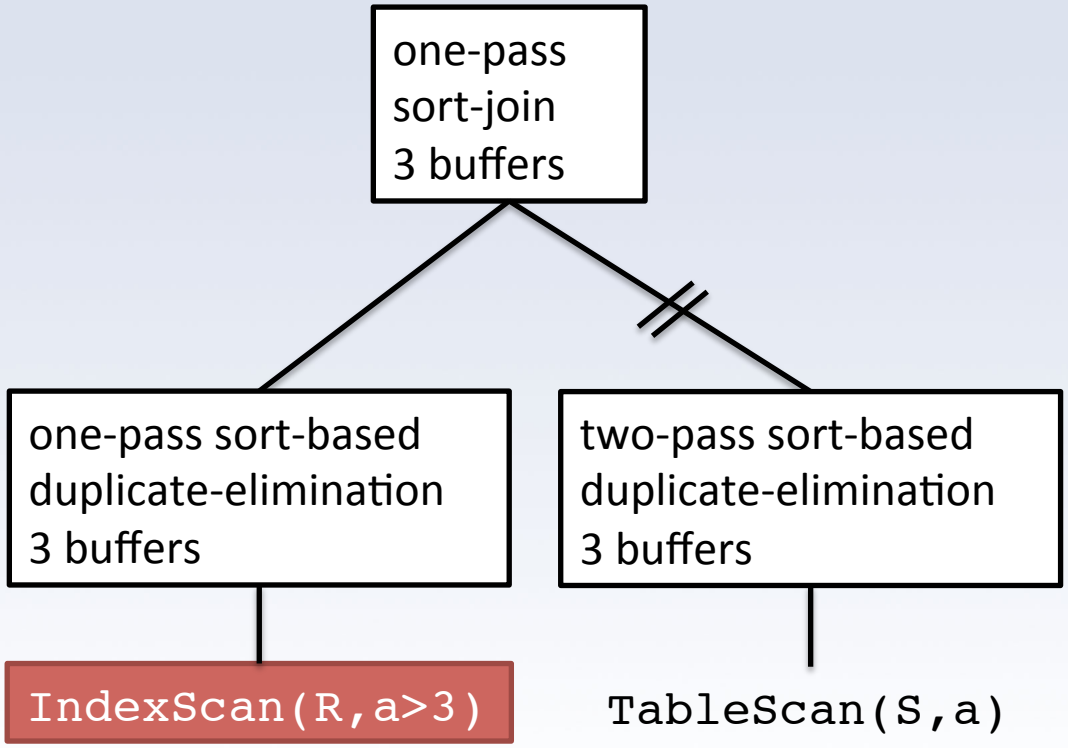

block 0
block 1
block 2
block 3
block 4
block 5
Q
X
F
A
R
T
V
L
O
Q
Y
F
K
B
S
C
Y
X
A
B
C
F
K
L
O
Q
R
S
T
V

Phase 2

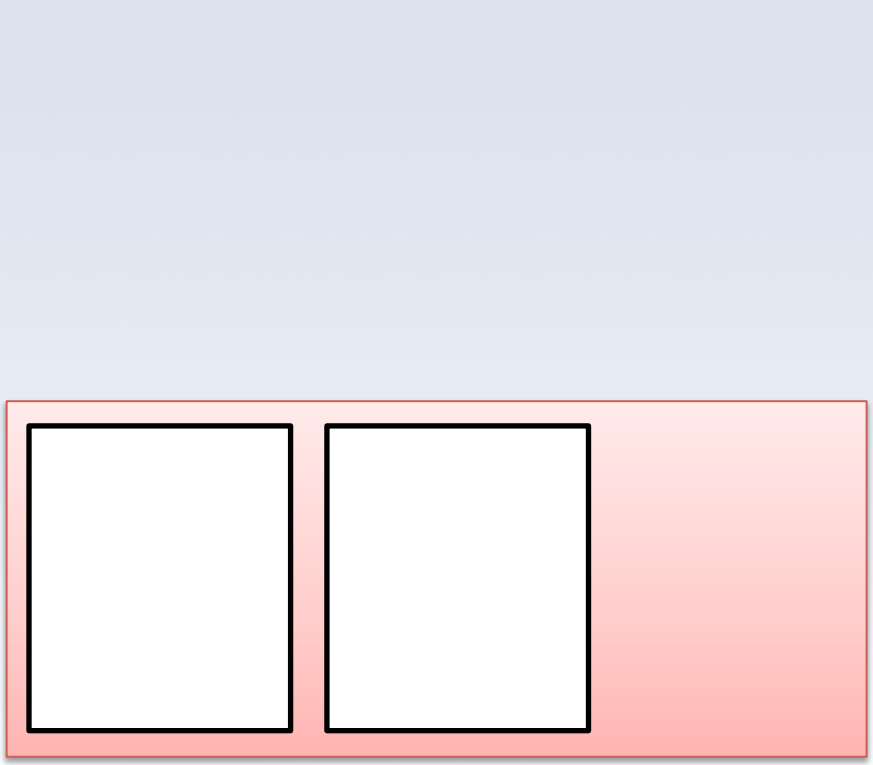

block 0
block 1
block 2
block 3
block 4
block 5
Q
X
F
A
R
T
V
L
O
Q
Y
F
K
B
S
C
X
Y
A
B
C
F
K
L
O
Q
R
S
T
V

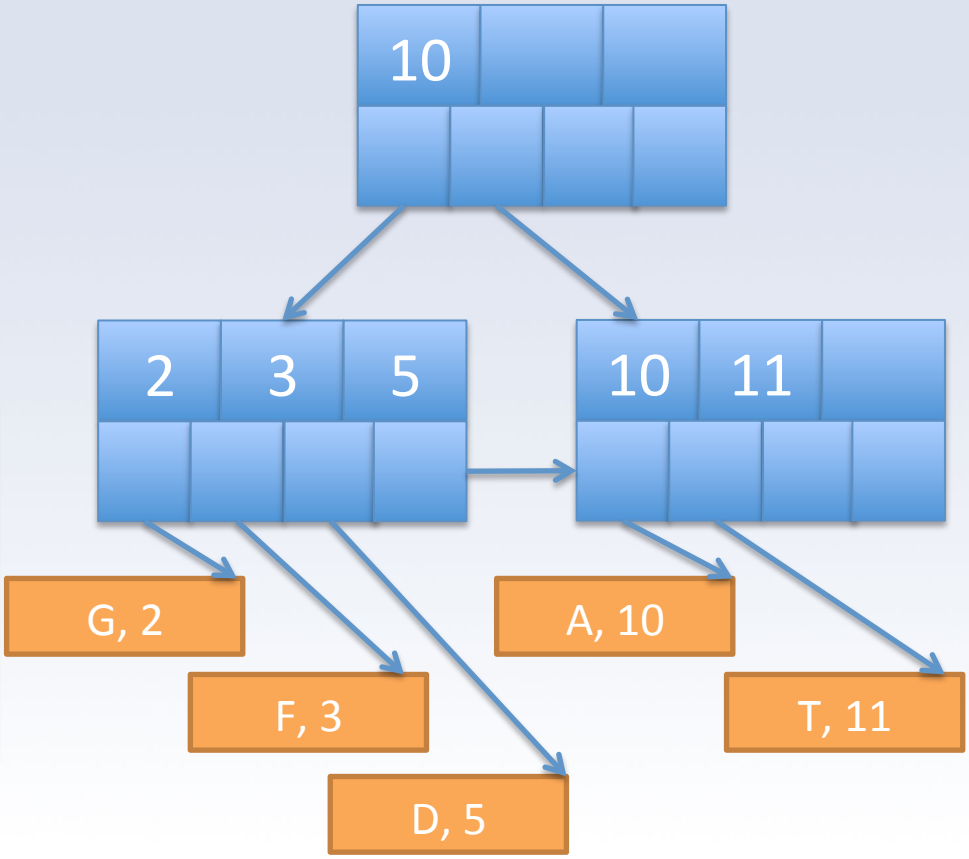
Phase 2


block 0
block 1
block 2
block 3
block 4
block 5
Q
X
F
A
R
T
V
L
O
Q
Y
F
K
B
S
C
A
B
C
F
K
L
O
Q
R
S
T
V
X
Y

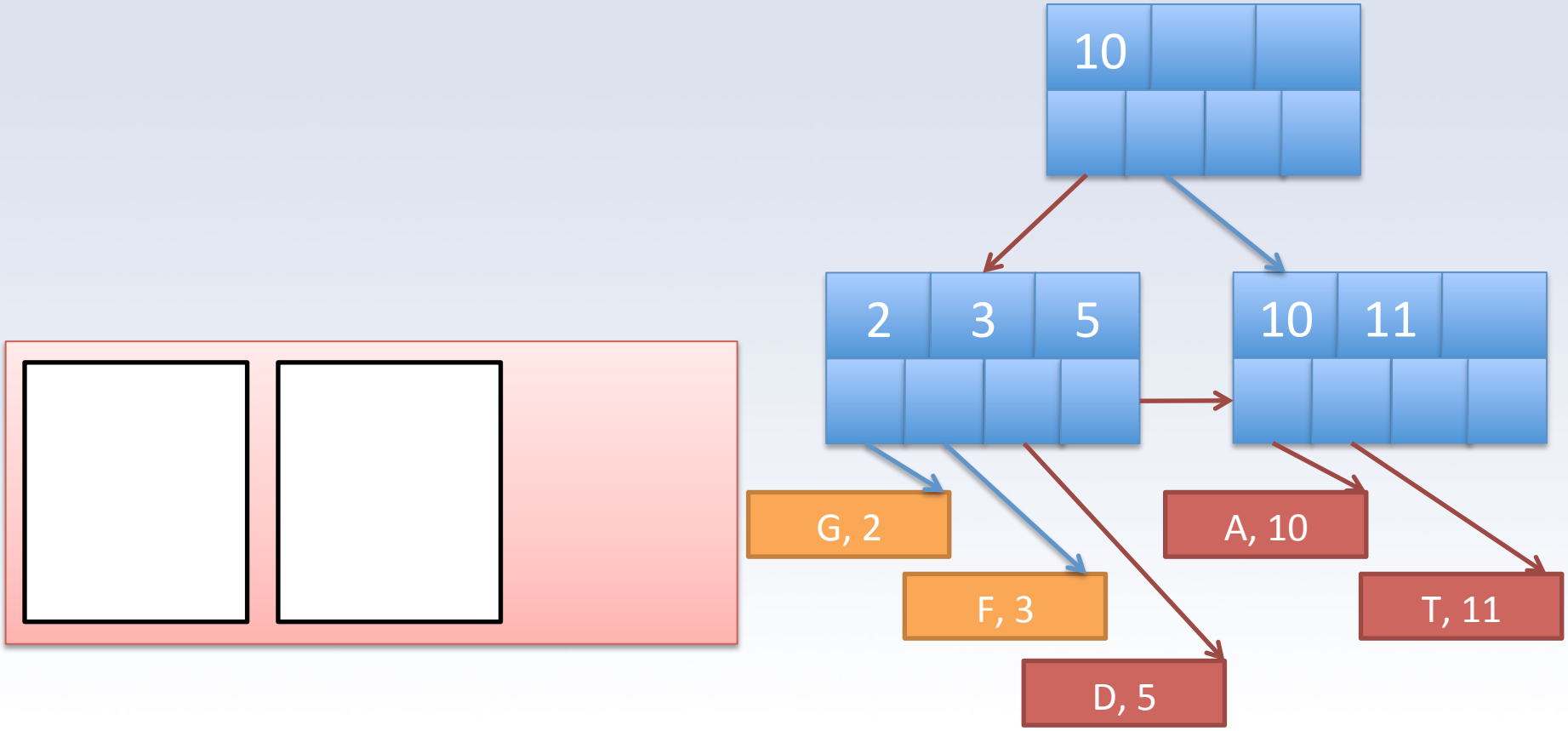

one-pass
sort-join
3 buffers
one-pass sort-based
duplicate-elimination
3 buffers
two-pass sort-based
duplicate-elimination
3 buffers
IndexScan(R,a>3)
TableScan(S,a)

Index Scan

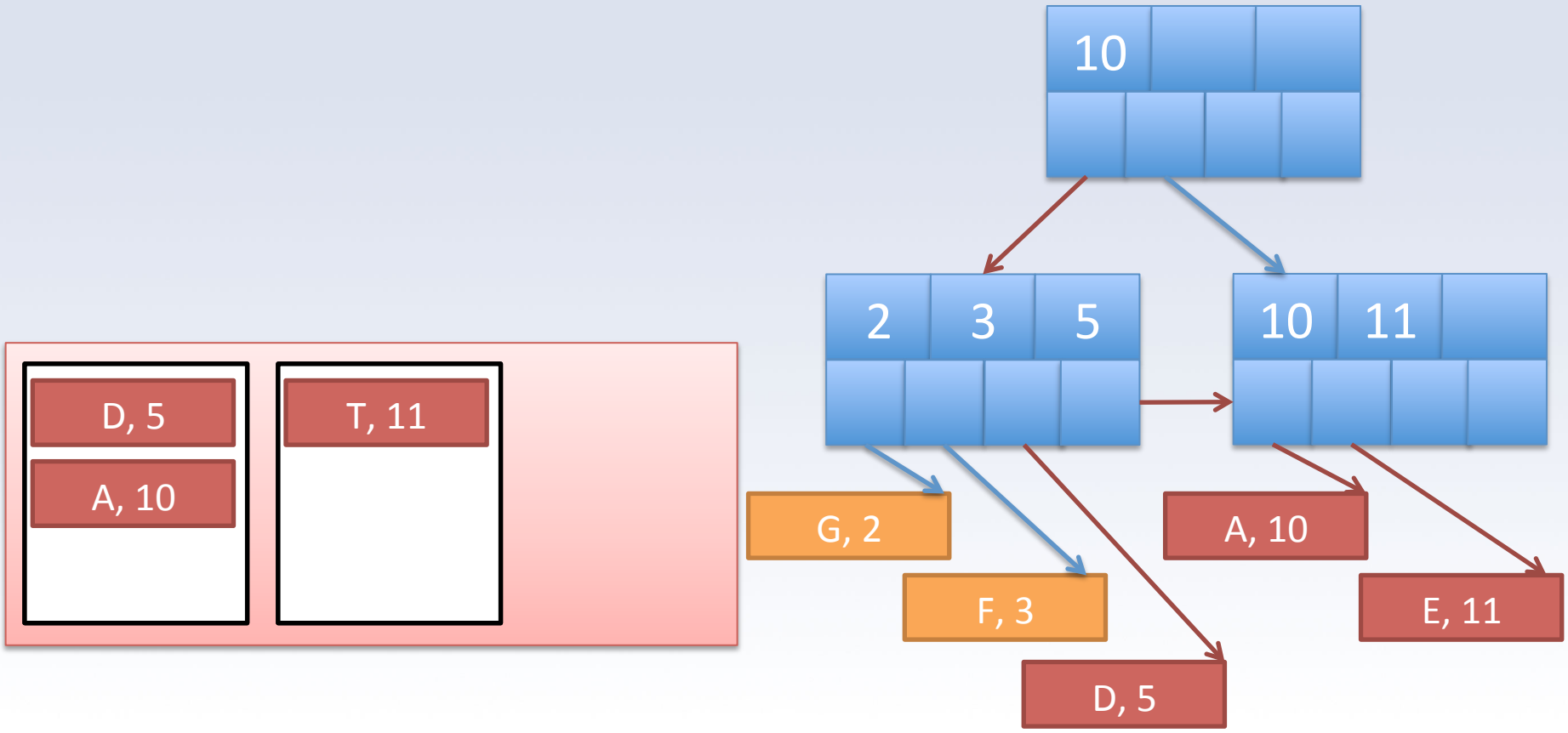


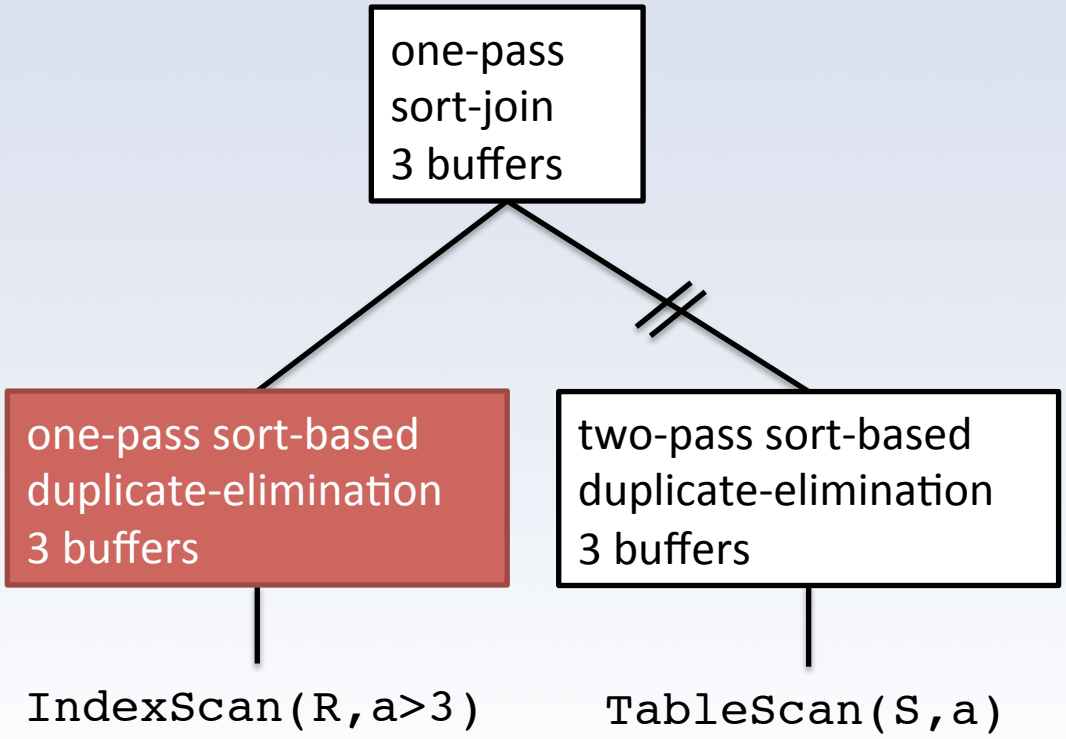


Index Scan

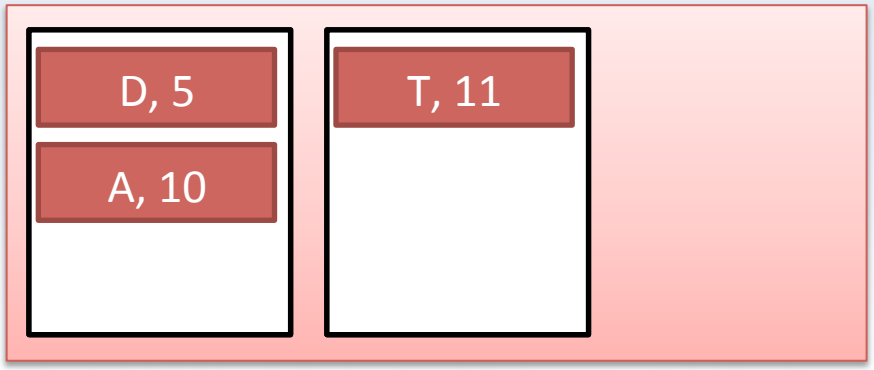


Index Scan

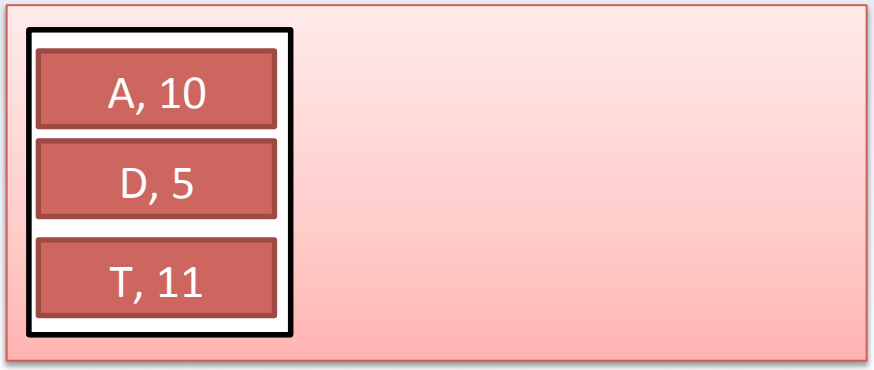



one-pass
sort-join
3 buffers
one-pass sort-based
duplicate-elimination
3 buffers
two-pass sort-based
duplicate-elimination
3 buffers
IndexScan(R,a>3)
TableScan(S,a)

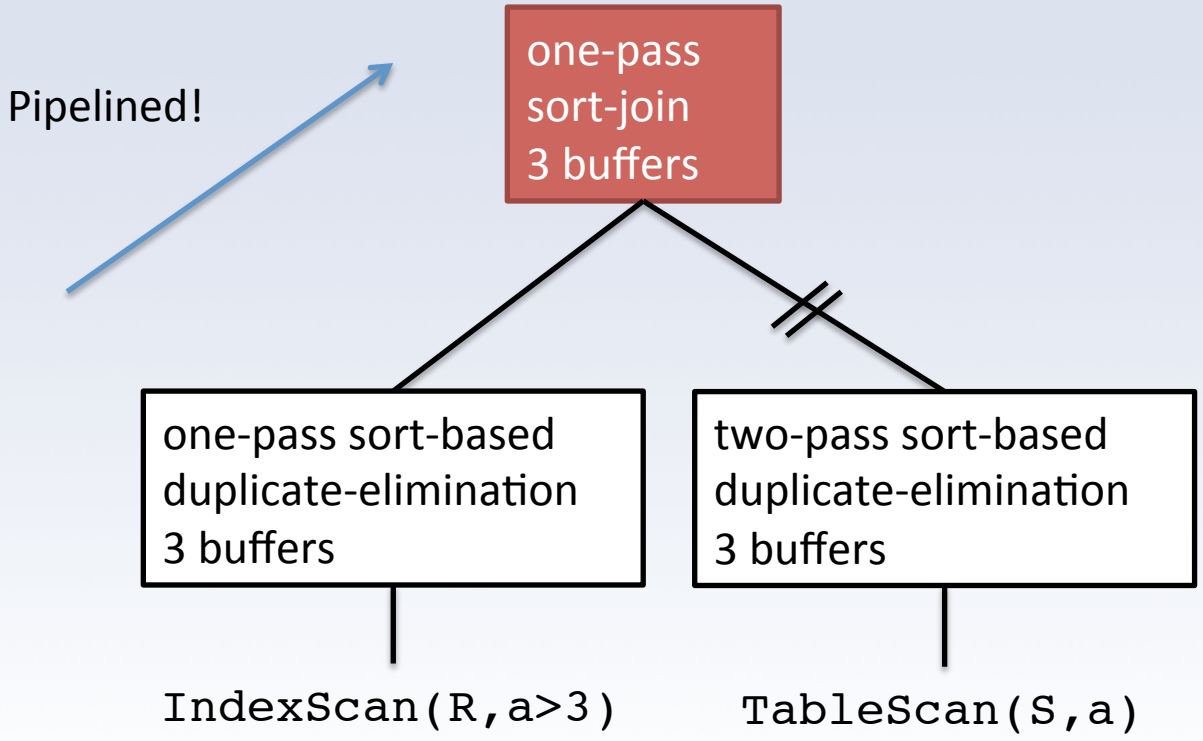
One Pass Sort



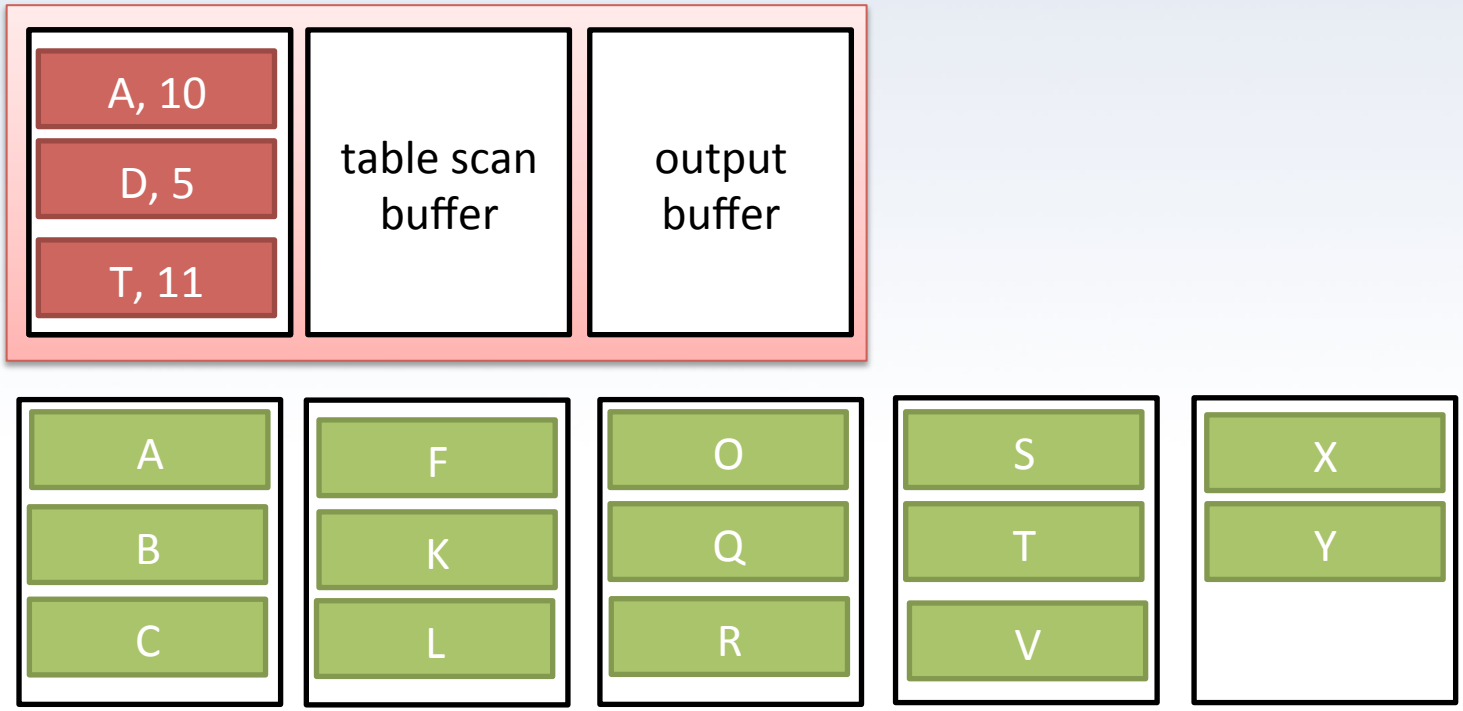
One Pass Sort



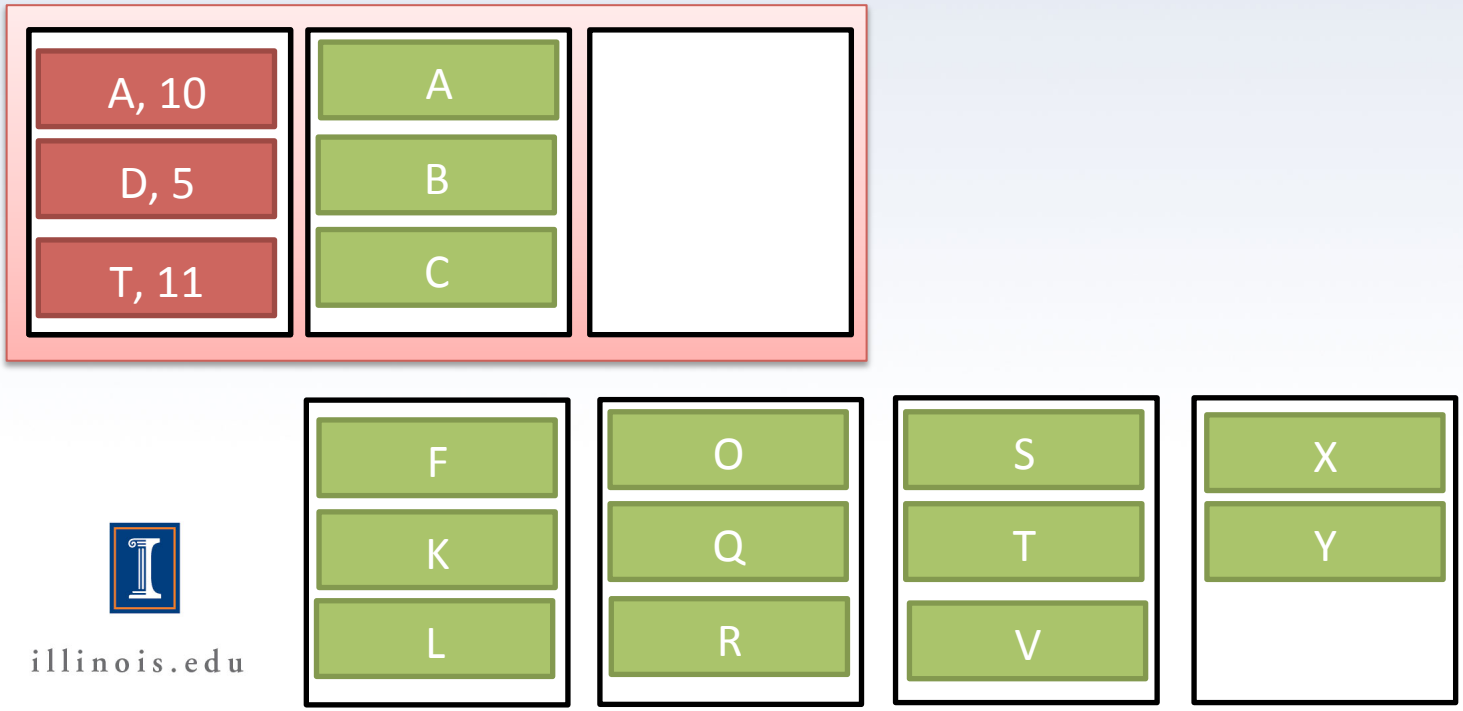
Physical Query Plan



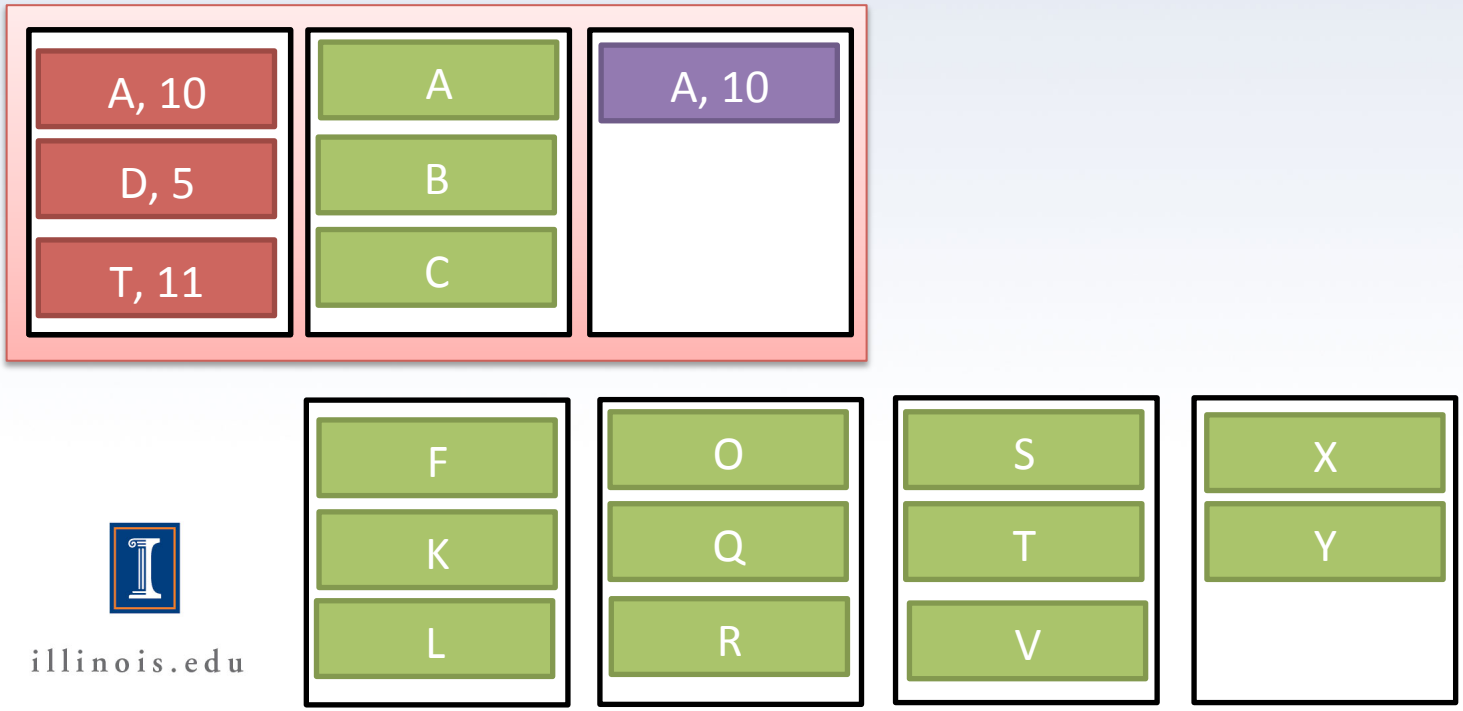
One Pass Sort-Join



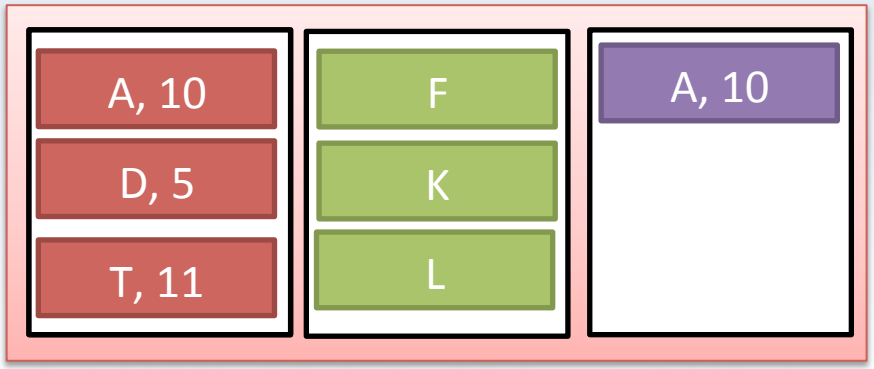
One Pass Sort-Join

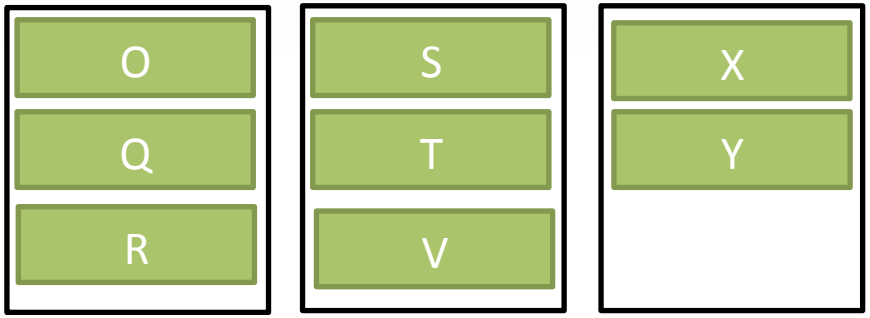


One Pass Sort-Join



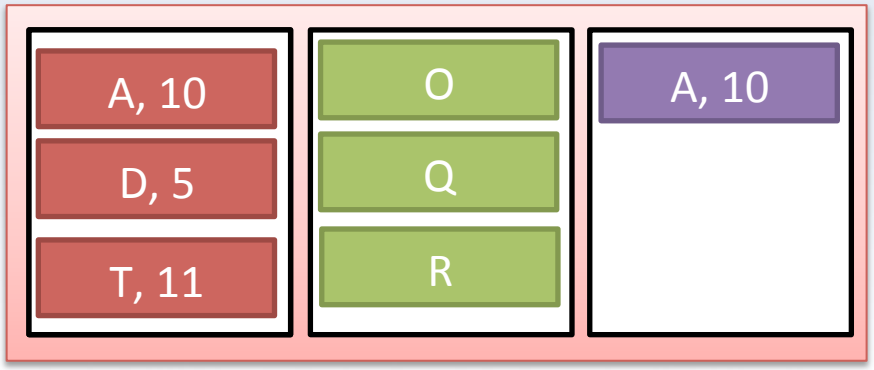
One Pass Sort-Join

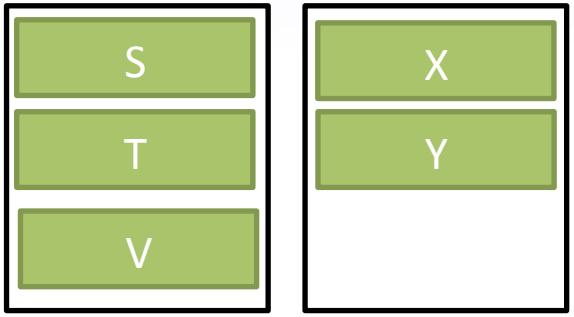




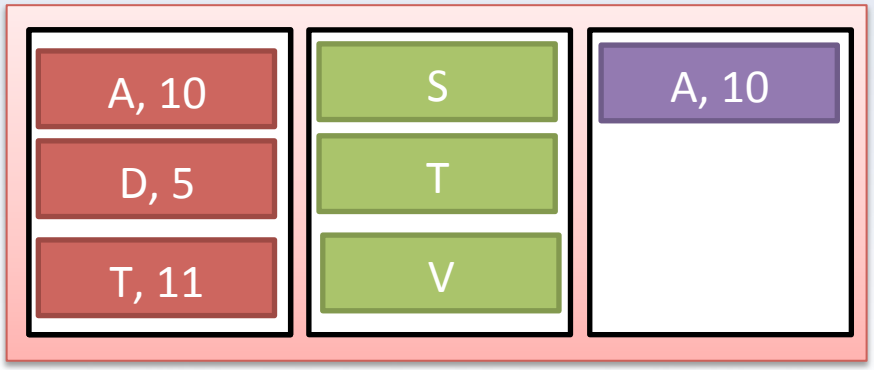
illinois.edu

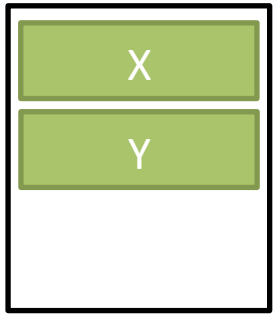
One Pass Sort-Join



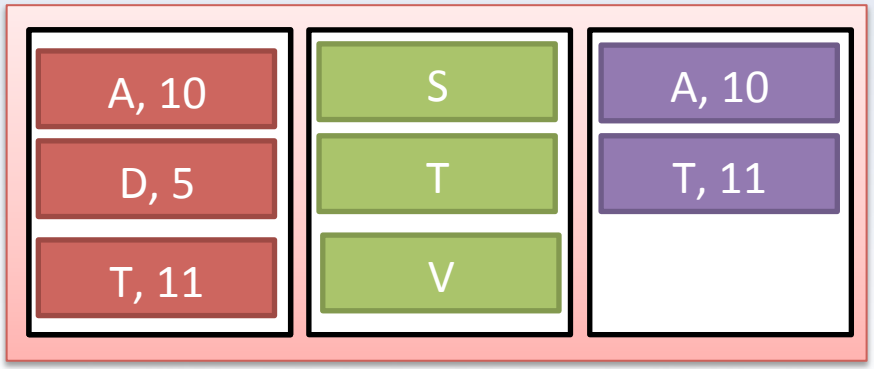


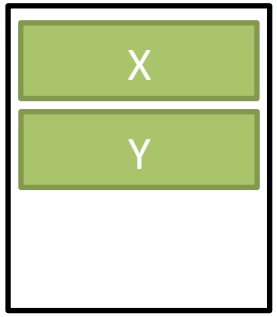
One Pass Sort-Join



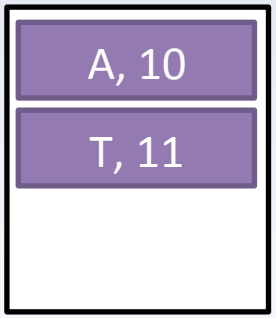


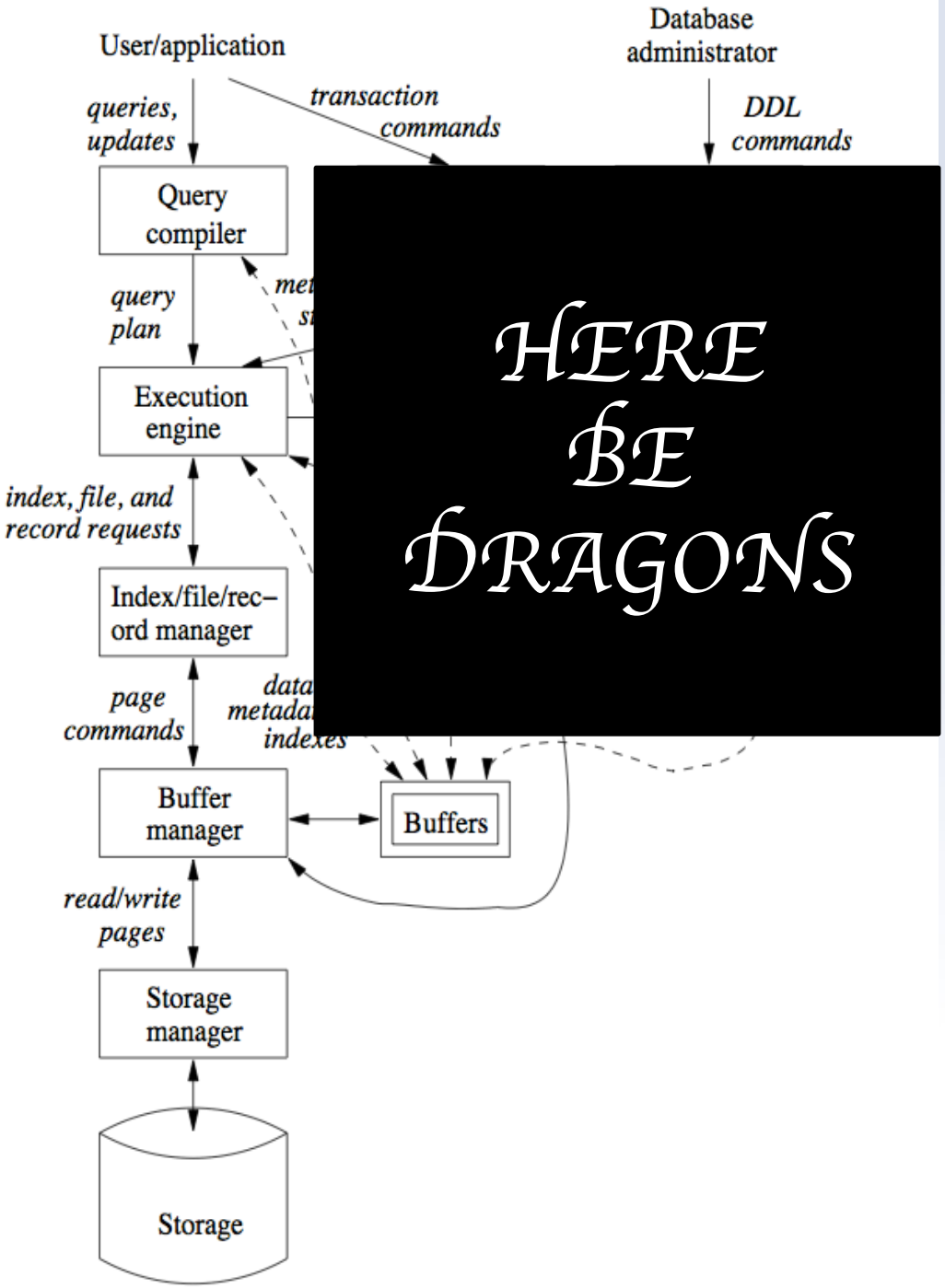
One Pass Sort-Join



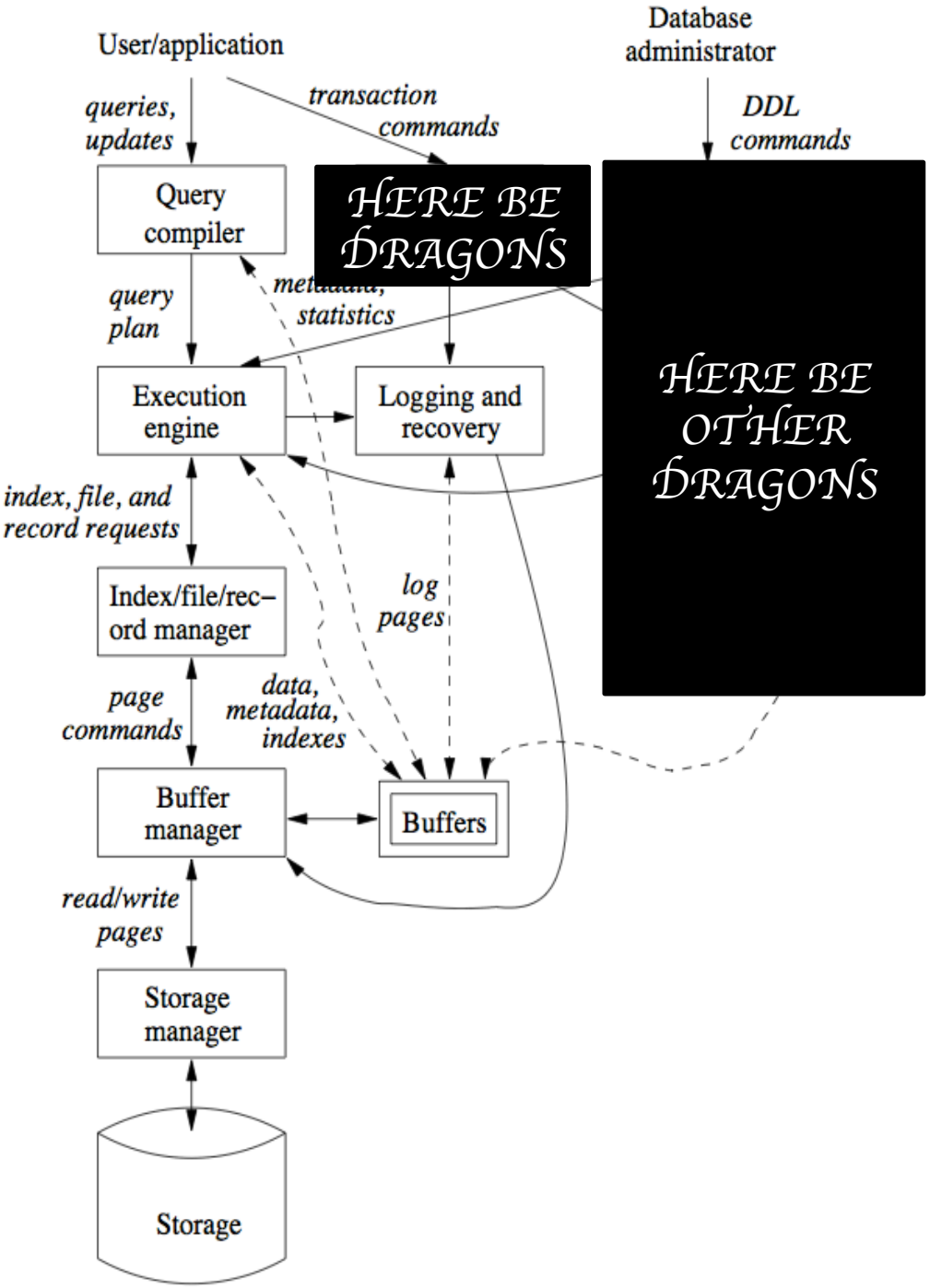


Output





illinois.edu


User/application
Database administrator
queries, updates
transaction commands
DDL commands
Query compiler
HERE BE DRAGONS
HERE BE OTHER DRAGONS
query plan
statistics
Execution engine
Logging and recovery
index, file, and record requests
log pages
Index/file/rec–ord manager
page commands
data, metadata, indexes
Buffer manager
Buffers
read/write pages
Storage manager
Storage

Logging and Recovery

- System failure or error: return the database to a valid state
  - transactions that are complete should be ***durable*** (written permanently to disk)
  - transactions that are incomplete should be ***aborted*** (appear to never have happened)

Logging and Recovery

- ACID requirements:
  - atomicity
  - consistency
  - isolation
  - durability

Logging and Recovery

- ACID requirements:
  - atomicity (Interruptions, ROLLBACK)
  - consistency (Constraints/Triggers)
  - isolation
  - durability

Transaction Primitives

- INPUT(X) - copy X into memory buffer
- READ(X,t) - assign X to variable t
- WRITE(X,t) - copy value of t into X
- OUTPUT(X) - write memory buffer containing X to disk

Example

Constraint: A=B

Transaction:

A=A*2

B=B*2

Example

Constraint: A=B

Transaction:

A=A*2

B=B*2

Action	t	Mem A	Mem B	Disk A	Disk B
READ(A,t)	8	8		8	8

Example

Constraint: A=B

Transaction:

A=A*2

B=B*2

Action	t	Mem A	Mem B	Disk A	Disk B
READ(A,t)	8	8		8	8
t=t*2	16	8		8	8

Example

Constraint: A=B

Transaction:

A=A*2

B=B*2

Action	t	Mem A	Mem B	Disk A	Disk B
READ(A,t)	8	8		8	8
t=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8

Example

Constraint: A=B

Transaction:

A=A*2

B=B*2

Action	t	Mem A	Mem B	Disk A	Disk B
READ(A,t)	8	8		8	8
t=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
READ(B,t)	8	16	8	8	8

Example

Constraint: A=B

Transaction:

A=A*2

B=B*2

Action	t	Mem A	Mem B	Disk A	Disk B
READ(A,t)	8	8		8	8
t=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
READ(B,t)	8	16	8	8	8
t=t*2	16	16	8	8	8

Example

Constraint: A=B

Transaction:

A=A*2

B=B*2

Action	t	Mem A	Mem B	Disk A	Disk B
READ(A,t)	8	8		8	8
t=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
READ(B,t)	8	16	8	8	8
t=t*2	16	16	8	8	8
WRITE(B,t)	16	16	16	8	8

Example

Constraint: A=B

Transaction:

A=A*2

B=B*2

Action	t	Mem A	Mem B	Disk A	Disk B
READ(A,t)	8	8		8	8
t=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
READ(B,t)	8	16	8	8	8
t=t*2	16	16	8	8	8
WRITE(B,t)	16	16	16	8	8
OUTPUT(A)	16	16	16	16	8

Example

Constraint: A=B

Transaction:

A=A*2

B=B*2

Action	t	Mem A	Mem B	Disk A	Disk B
READ(A,t)	8	8		8	8
t=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
READ(B,t)	8	16	8	8	8
t=t*2	16	16	8	8	8
WRITE(B,t)	16	16	16	8	8
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16

Undo Logging

- Main idea: keep track of all values we change
- When recovering, restore the old values for incomplete transactions

Logging

- Sequentially write entries to disk:
  - <START T> - transaction started
  - <COMMIT T> - transaction successful
  - <ABORT T> - transaction unsuccessful
  - <T,X,v> - transaction changed element X from value v to some new value

Undo Logging

- Rules:
  1. Add <T,X,v> to log ***before*** new value of X is written to disk
  2. Add <COMMIT T> ***after all*** results for T have been written to disk

Example

Action	t	Mem A	Mem B	Disk A	Disk B	Log
						<START T>
READ(A,t)	8	8		8	8	
t=t*2	16	8		8	8	
WRITE(A,t)	16	16		8	8	<T,A,8>
READ(B,t)	8	16	8	8	8	
t=t*2	16	16	8	8	8	
WRITE(B,t)	16	16	16	8	8	<T,B,8>
FLUSH LOG						
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	
						<COMMIT T>
FLUSH LOG						

Recovery

- If we see <START T> without corresponding <COMMIT T>, must undo the transaction
- When the system fails, scan log backward
  - If we see <COMMIT F>, ignore transaction F
  - For every other transaction T, when we see <T,X,v>, write v to X

Example

<START T_1>

<T_1,A,5>

<T_1,B,3>

<START T_2>

<T_1,C,7>

<T_2,X,5>

<COMMIT T_2>

<START T_3>

<T_1,D,1>

<T_3,A,3>

<T_3,B,2>

<START T_4>

<COMMIT T_3>

<T_4,X,3>

Next time…

- We'll look at
  - Ways to avoid scanning entire log
  - Redo logging
  - Undo/Redo logging
  - Introduce transaction management