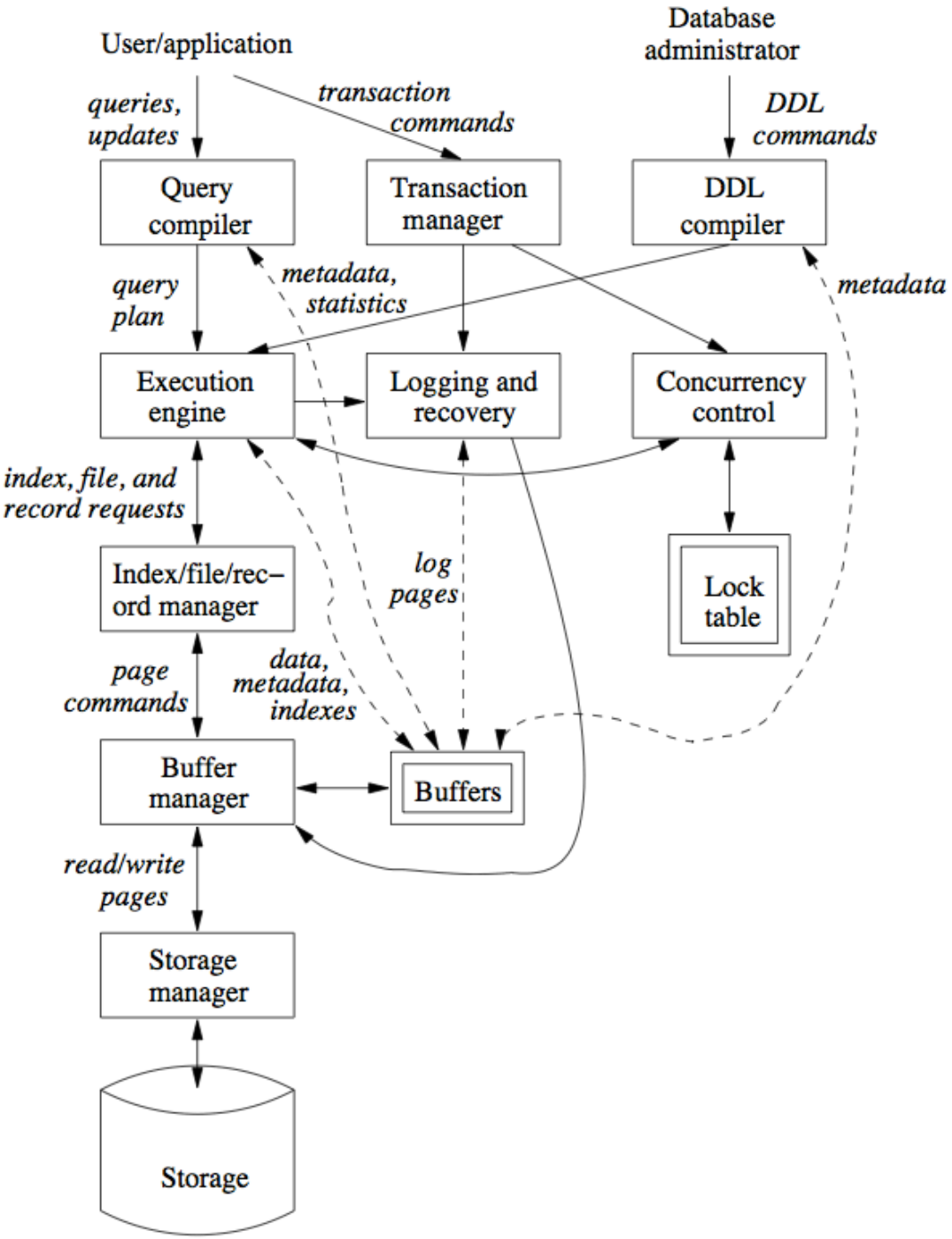
UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

CS411 - Query Execution

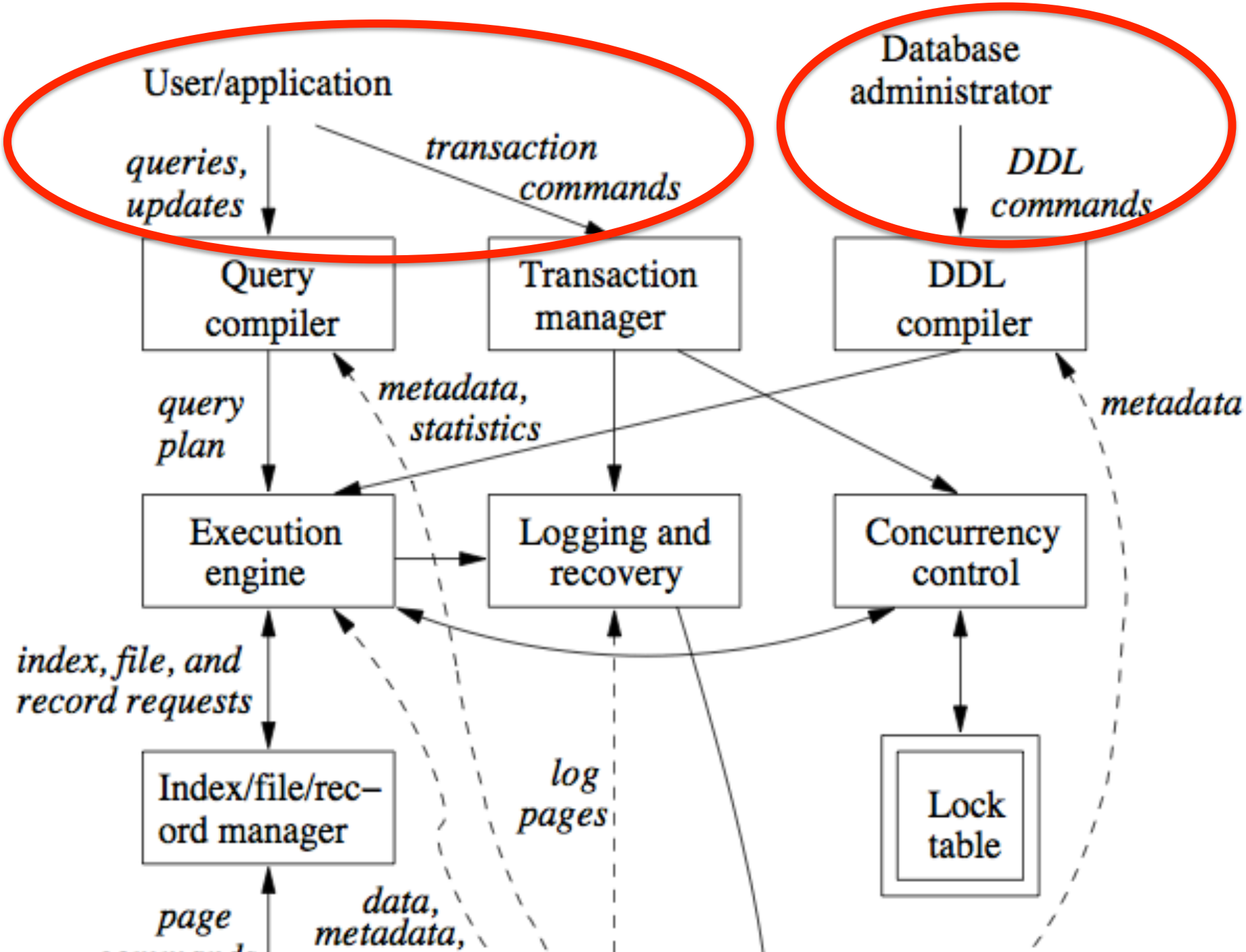
illinois.edu

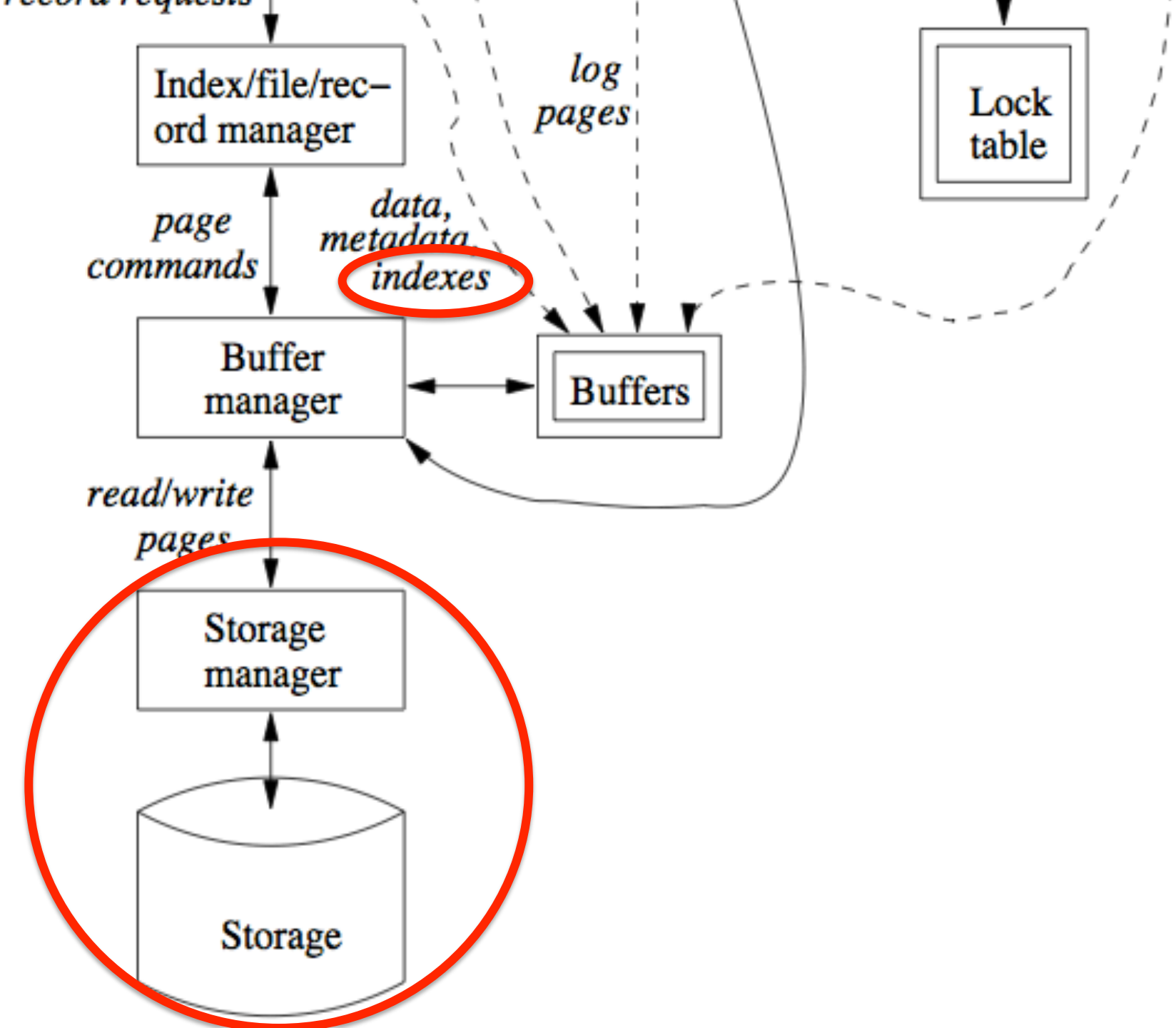


illinois.edu

So far, we've seen

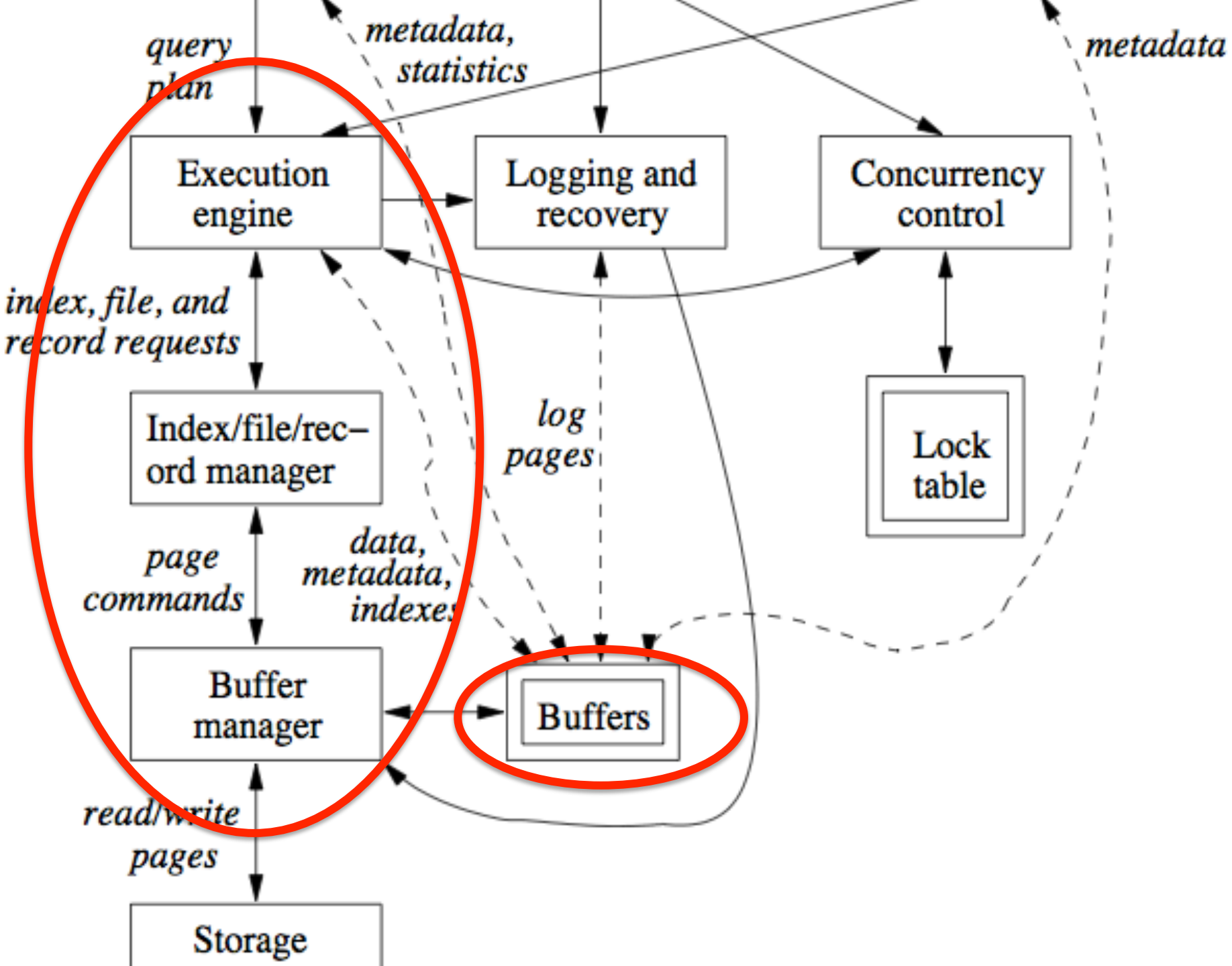
- How users/admins interact with a DBMS
- How records and relations are stored
- How to create structures to access records more quickly


User/application
queries, updates
transaction commands
Database administrator
DDL commands
Query compiler
Transaction manager
DDL compiler
query plan
metadata, statistics
metadata
Execution engine
Logging and recovery
Concurrency control
index, file, and record requests
Index/file/rec–ord manager
log pages
Lock table
page
data, metadata,


Index/file/rec–
ord manager
log
pages
Lock
table
page
commands
data,
metadata,
indexes
Buffer
manager
Buffers
read/write
pages
Storage
manager
Storage

Next, we will see

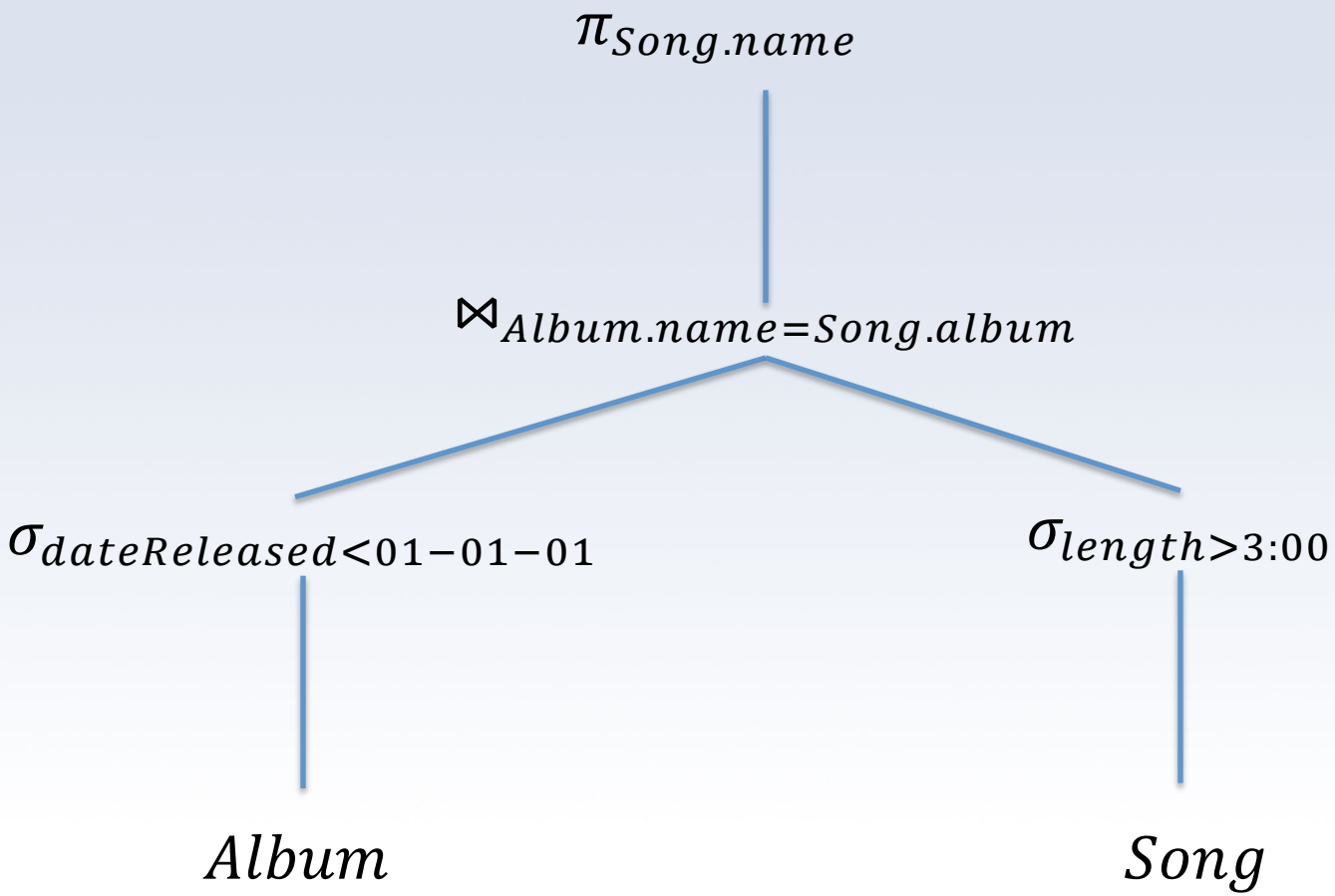
- Many ways to execute query operations on our data
  - Today: without using indexes
  - Friday: taking advantage of indexes
- Next week:
  - How user queries become query plans
  - How we can efficiently execute query plan


query plan
metadata, statistics
metadata
Execution engine
Logging and recovery
Concurrency control
index, file, and record requests
Index/file/rec–ord manager
log pages
Lock table
page commands
data, metadata, indexes
Buffer manager
Buffers
read/write pages
Storage

Query Compilation

1. Parsing - the query is transformed into a parse tree
2. Parse tree converted into a logical query plan
3. Logical query plan is converted into a physical query plan

Query Plan


$\pi_{Song.name}$
$\bowtie_{Album.name=Song.album}$
$\sigma_{dateReleased<01-01-01}$
$\sigma_{length>3:00}$
Album
Song

Query Compilation

1. Parsing - the query is transformed into a parse tree
2. Parse tree converted into a logical query plan
3. Logical query plan is converted into a physical query plan

Physical Query Plan

- Given a tree of relational algebra operators, how can we physically execute them?
- How do we access the data efficiently to compute the desired result?

Review

- Let’s take a quick step back

Disk access

- Fundamental unit of transfer is a *block*
- Blocks are typically 4-64kb
- When in main memory, blocks are sometimes called *pages*

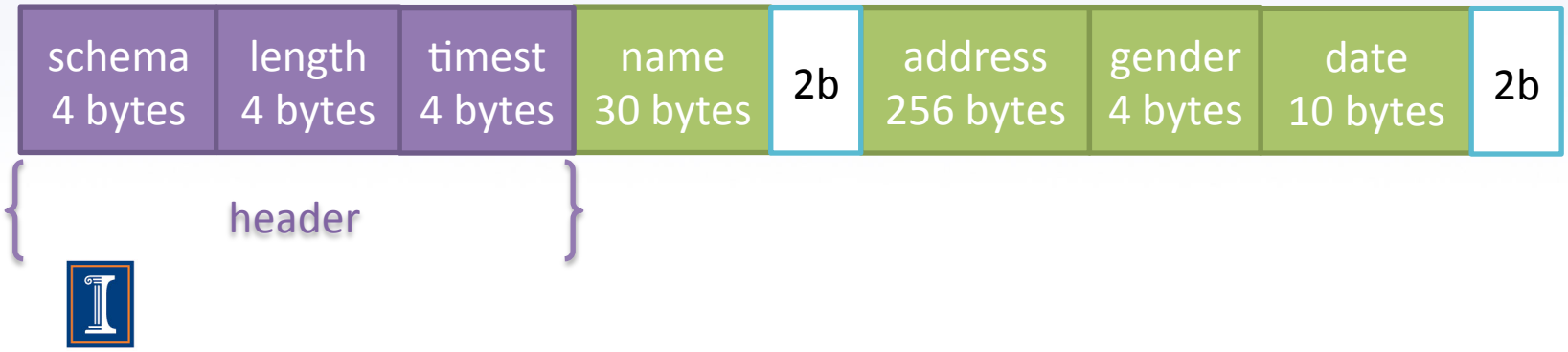
Fixed Length Records

- Align fields along ***word boundaries***
  - minimum unit addressable in memory
  - usually 4 or 8 bytes
- Need to include a header
  - pointer to the schema for the record
  - length of the record
  - last modified timestamp/access metadata
  - pointers to the fields

Fixed Length Records

```
CREATE TABLE Person(
    name CHAR(30) PRIMARY KEY,
    address VARCHAR(255),
    gender CHAR(1),
    birthdate DATE
);
```

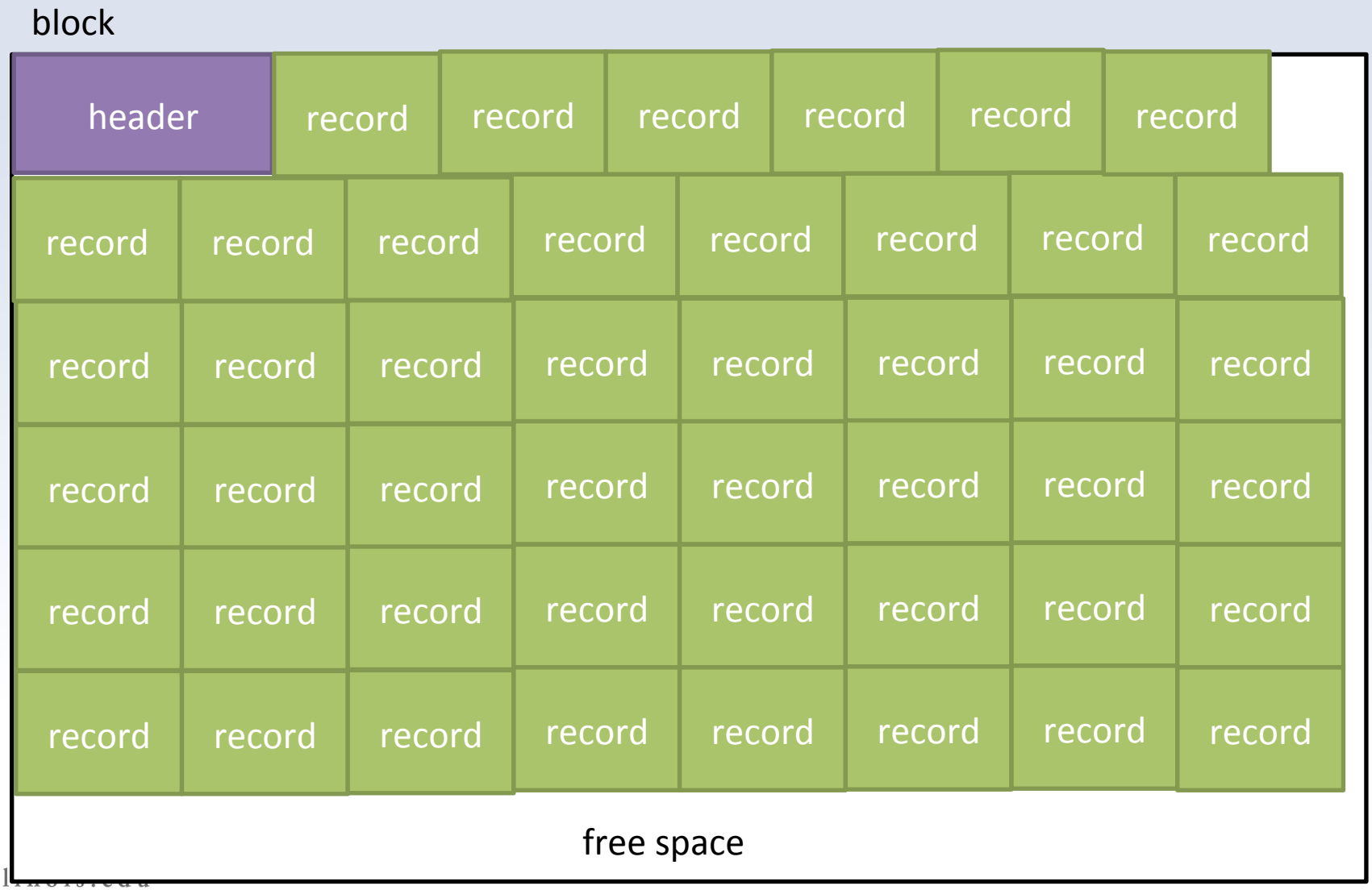
Actually takes 316 bytes



Putting records into blocks

- Doesn't necessarily fill the block
- Has a header:
  - Information about the relation
  - "Directory" of records in the block
  - Links to the "previous" and "next" blocks
  - Last modified timestamp/access metadata

Putting records into blocks

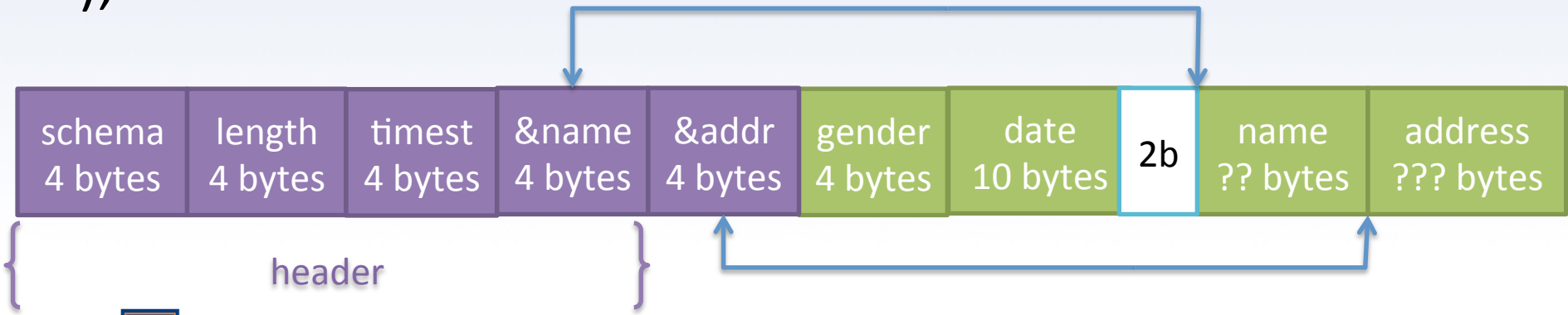


Variable length records

- Sometimes, fields have varying lengths
  - e.g. name, address
- Solution:
  - store fixed length fields first
  - store pointers to beginnings of variable length fields in header

Variable length records

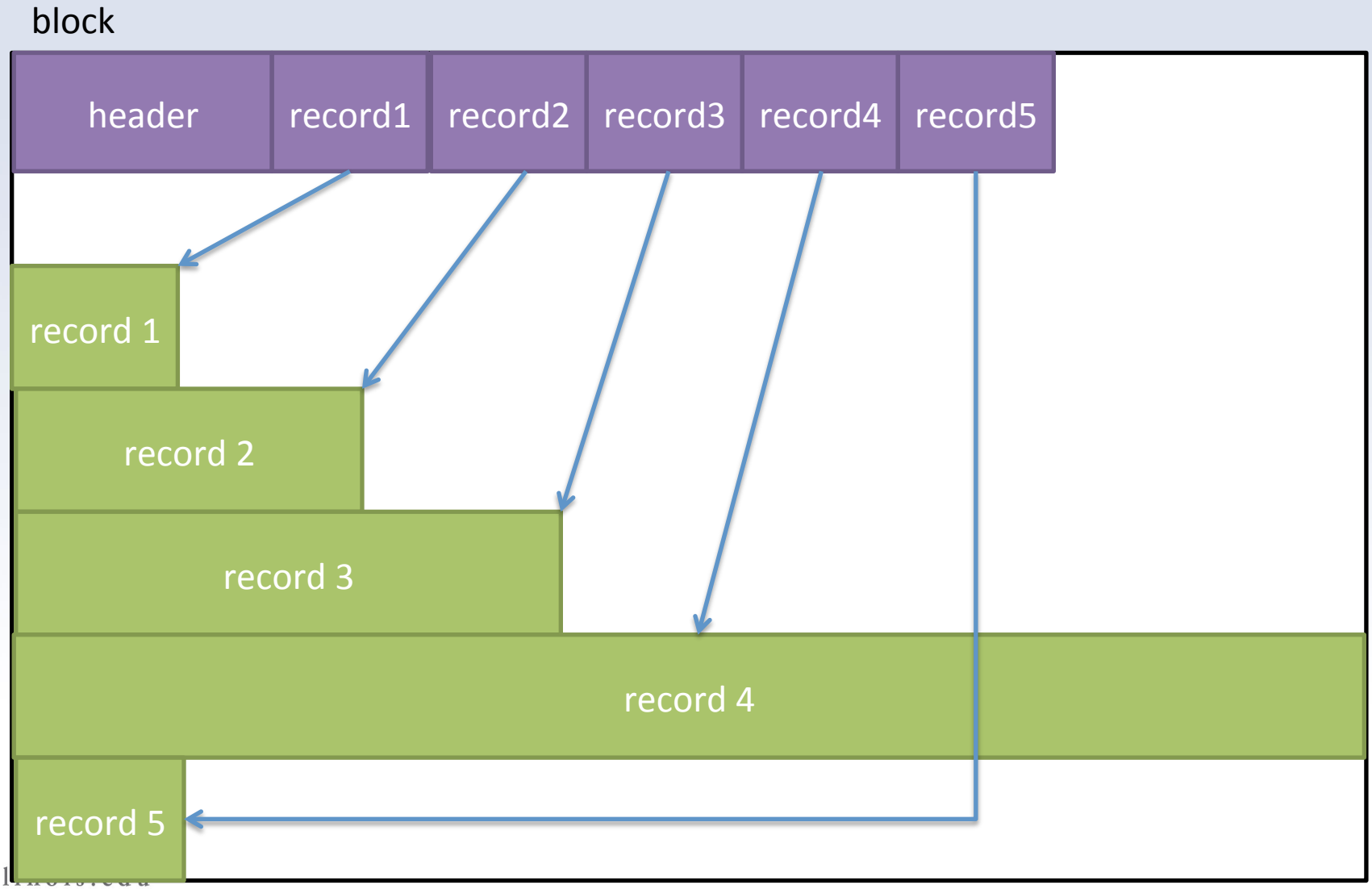
```
CREATE TABLE Person(
    name VARCHAR(30) PRIMARY KEY,
    address VARCHAR(255),
    gender CHAR(1),
    birthdate DATE
);
```



Variable length into blocks

- Added offset and length index to header for variable length records

Variable length into blocks



Big picture

- To access a relation, we pull the corresponding blocks into memory
- With a block in memory, we can access all of the records in that block

Query Execution

- Implement relational operators on our stored records
- Will implement in many different ways
  - Depending on the size of memory and our data, some methods better than others
  - Gives flexibility for query optimization

Goal

- We care about these operators:
  - σ selection and π projection
  - δ duplicate-elimination and γ grouping
  - ∪ union, ∩ intersection, and - difference
  - ⋈ join and × cross product
- Always assume relations are ***bags***

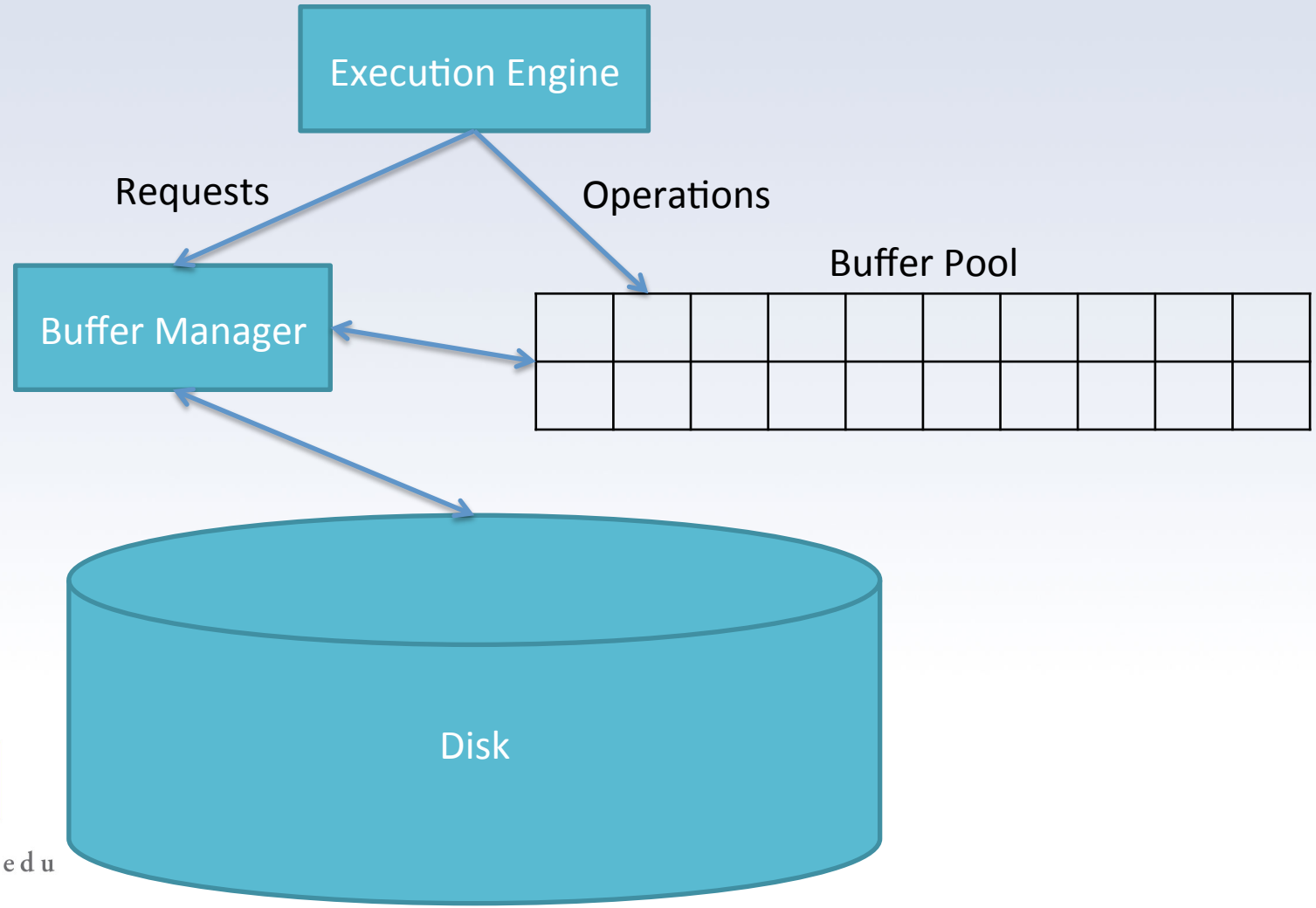
Three types of operators

- Tuple-at-a-time unary - need one tuple to perform computation (e.g. σ and π)
- Full-relational unary - need entire relation to perform computation (e.g. δ and γ)
- Full-relational binary - need two entire relations to perform computation (e.g. ∪, ∩, -, ⋈, ×)

Buffer Management

- We divide memory up into block sized *buffers*
- The *buffer manager* is responsible for allocating buffers for query execution
- Query execution brings blocks into and out of the *buffer pool* for processing

Buffer management



Iterator Model

- Operators can all be implemented with three functions:
  - `Open()`: initializes the data structures
  - `GetNext()`: finds the next tuple
  - `Close()`: clean up
- Easy to combine operations
- Enables pipelining

Measuring Cost of Operation

- We only care about I/O operations
- We also ignore output (write) costs:
  - Assume we can pipeline operations
  - Result of query is independent of execution strategy

Measuring Cost of Operation

- M - the number of blocks that fit in memory (called *buffers*)
- Given a relation R
  - B(R) - number of blocks to store R
  - T(R) - number of tuples in R
  - V(R,a) - number of distinct values for attribute a in relation R

Table Scan

- Pull records of a relation into memory one at a time
- Important operator not in relational algebra
- Example:

```
SELECT * FROM R
```

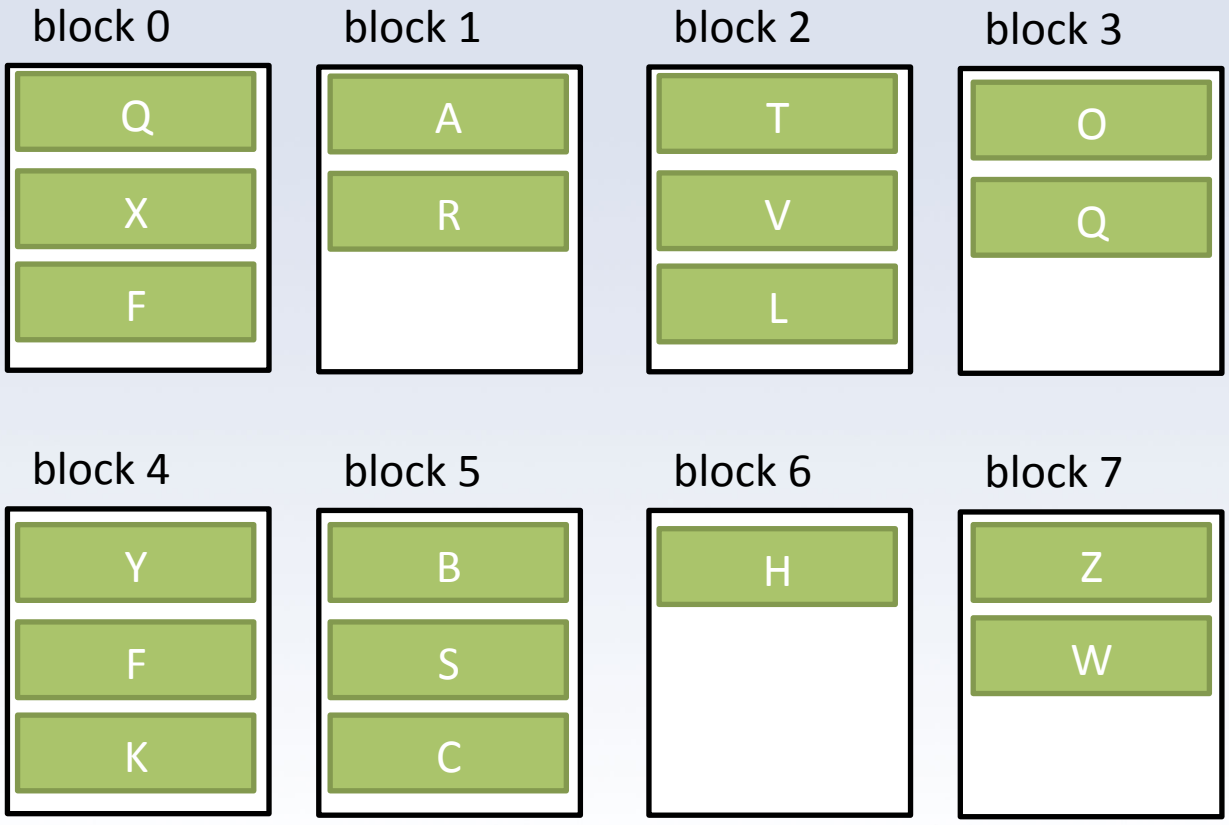
Table Scan

```
void open(){
   block* b=R.first_block;
   record* t=b.first_tuple;
}
```

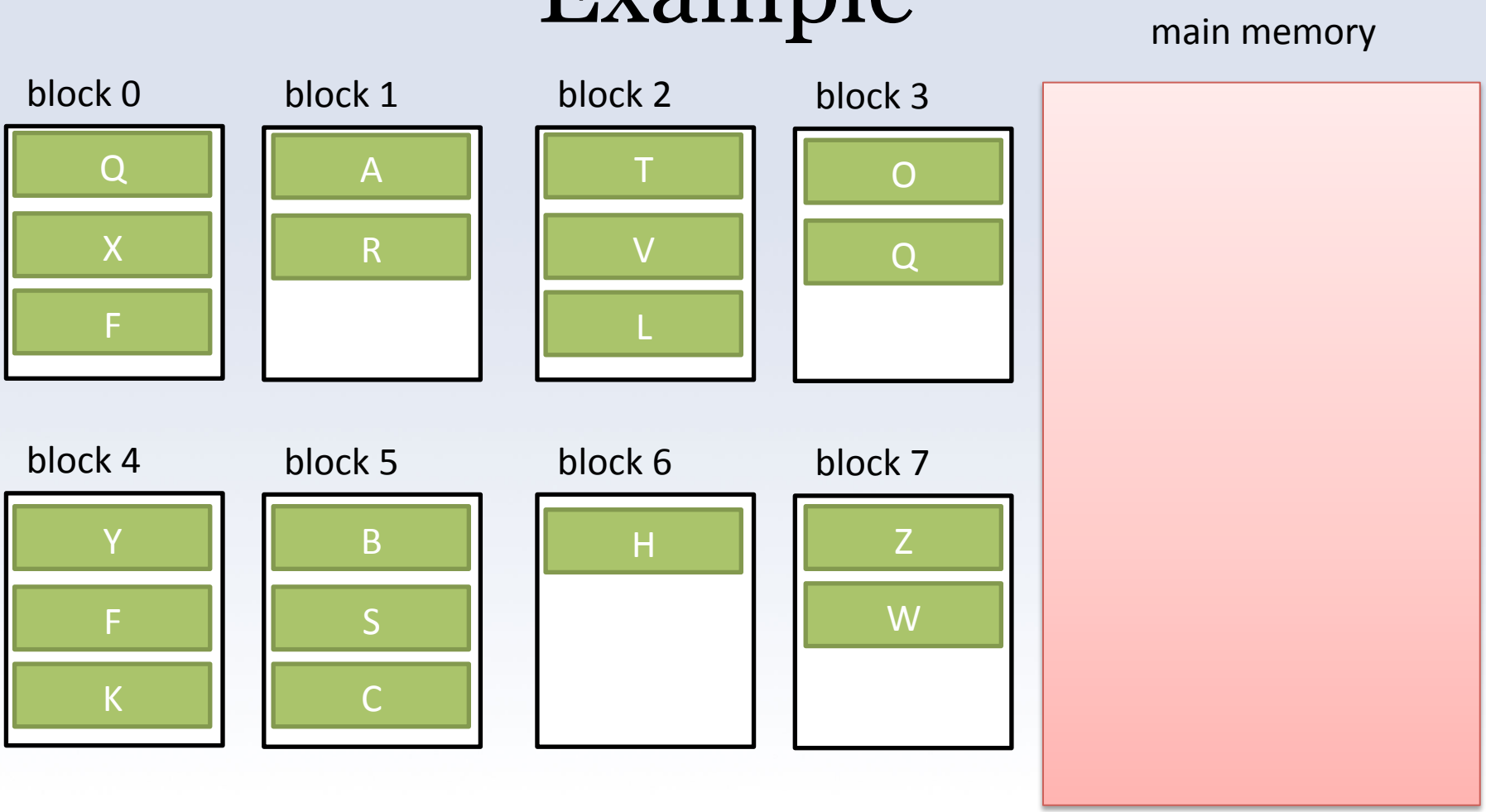
Table Scan

```
record* getNext(){
   if(t==NULL){
      b=b.next_block;
      if(b==NULL){return NULL;}
      t=b.first_tuple;
   }
   record* old_t=t;
   t+=(sizeof(record);
   return old_t;
}
```

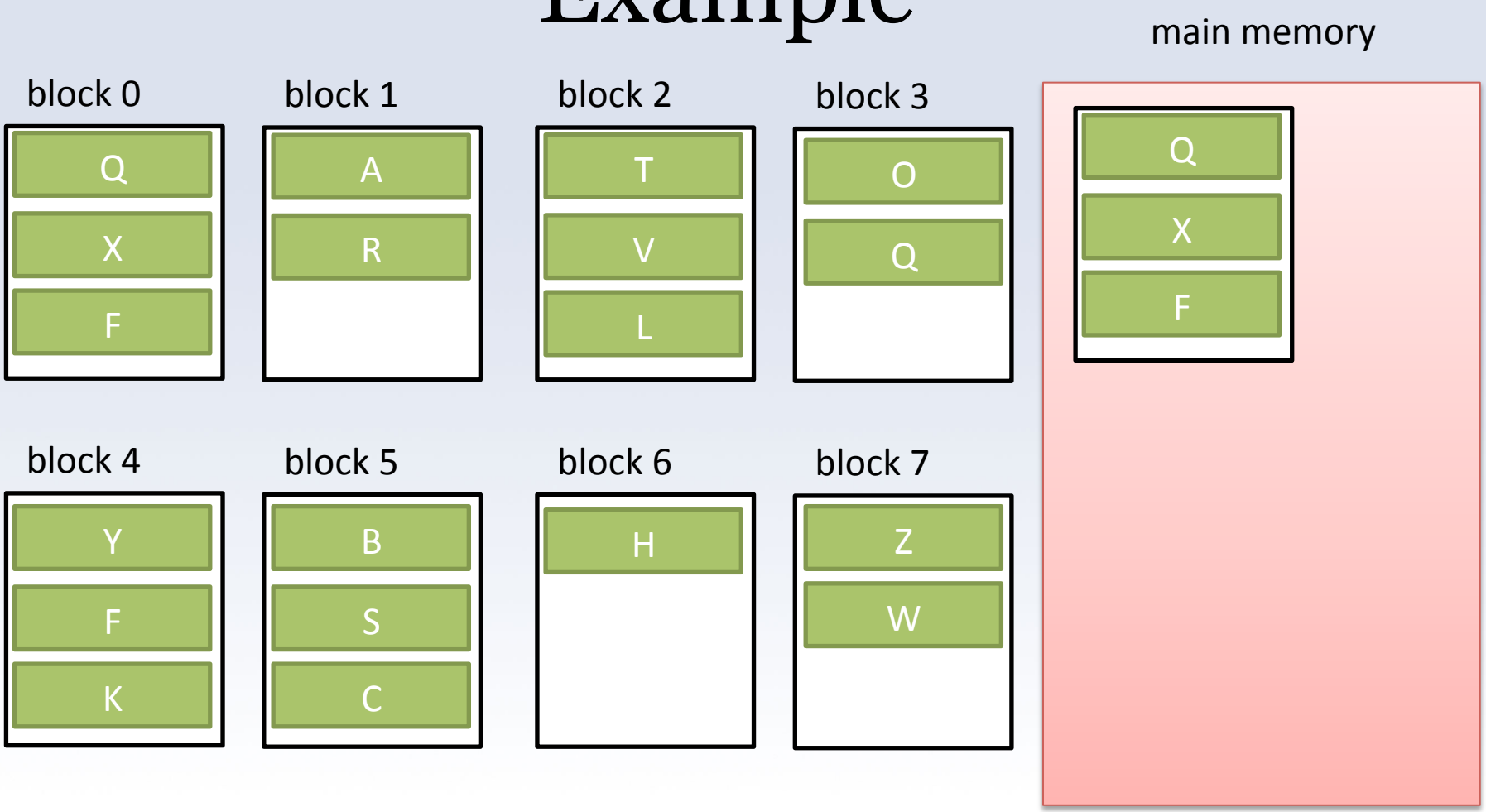
Example


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
block 6
H
block 7
Z
W

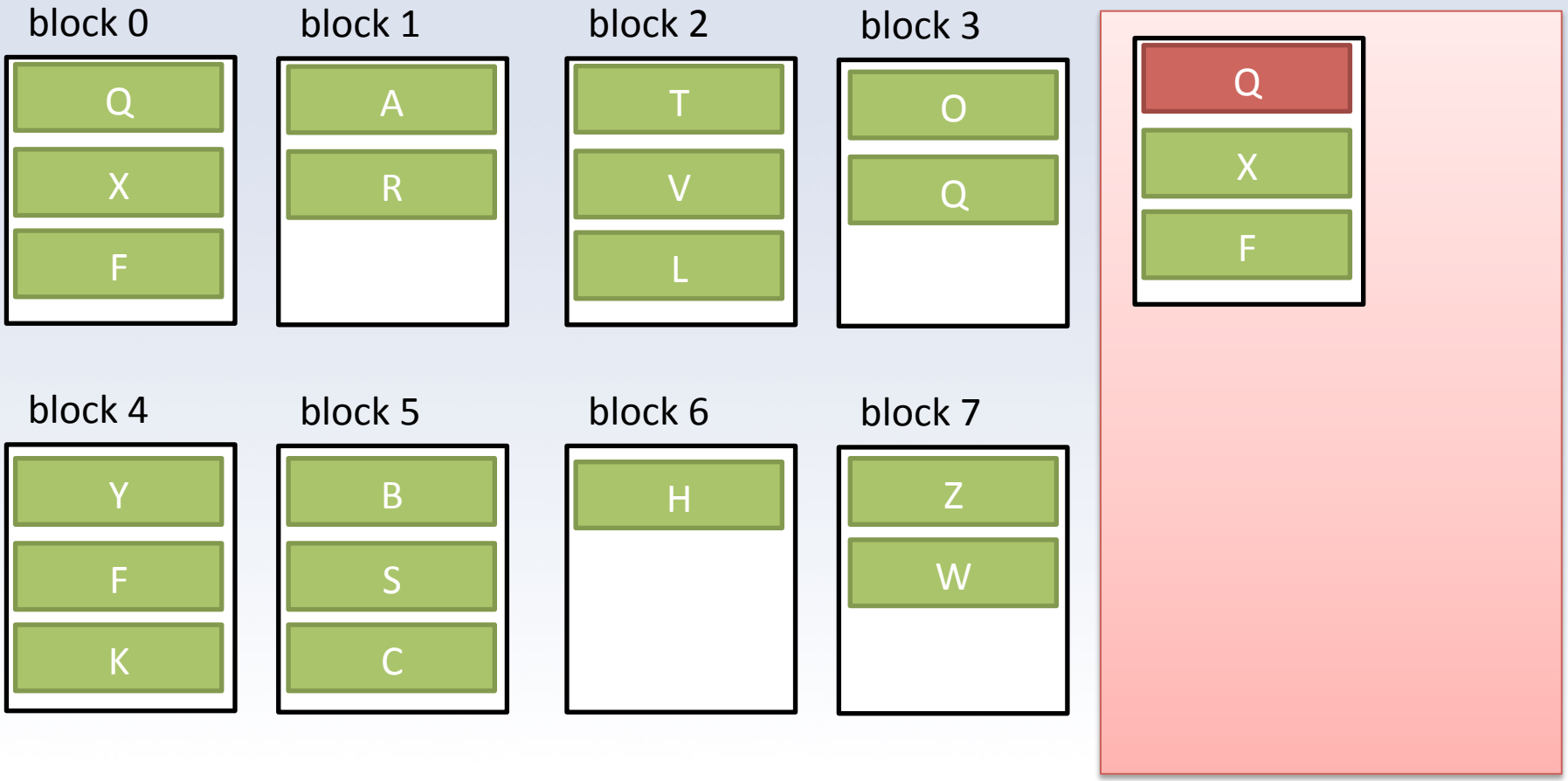
Example


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
block 6
H
block 7
Z
W
main memory

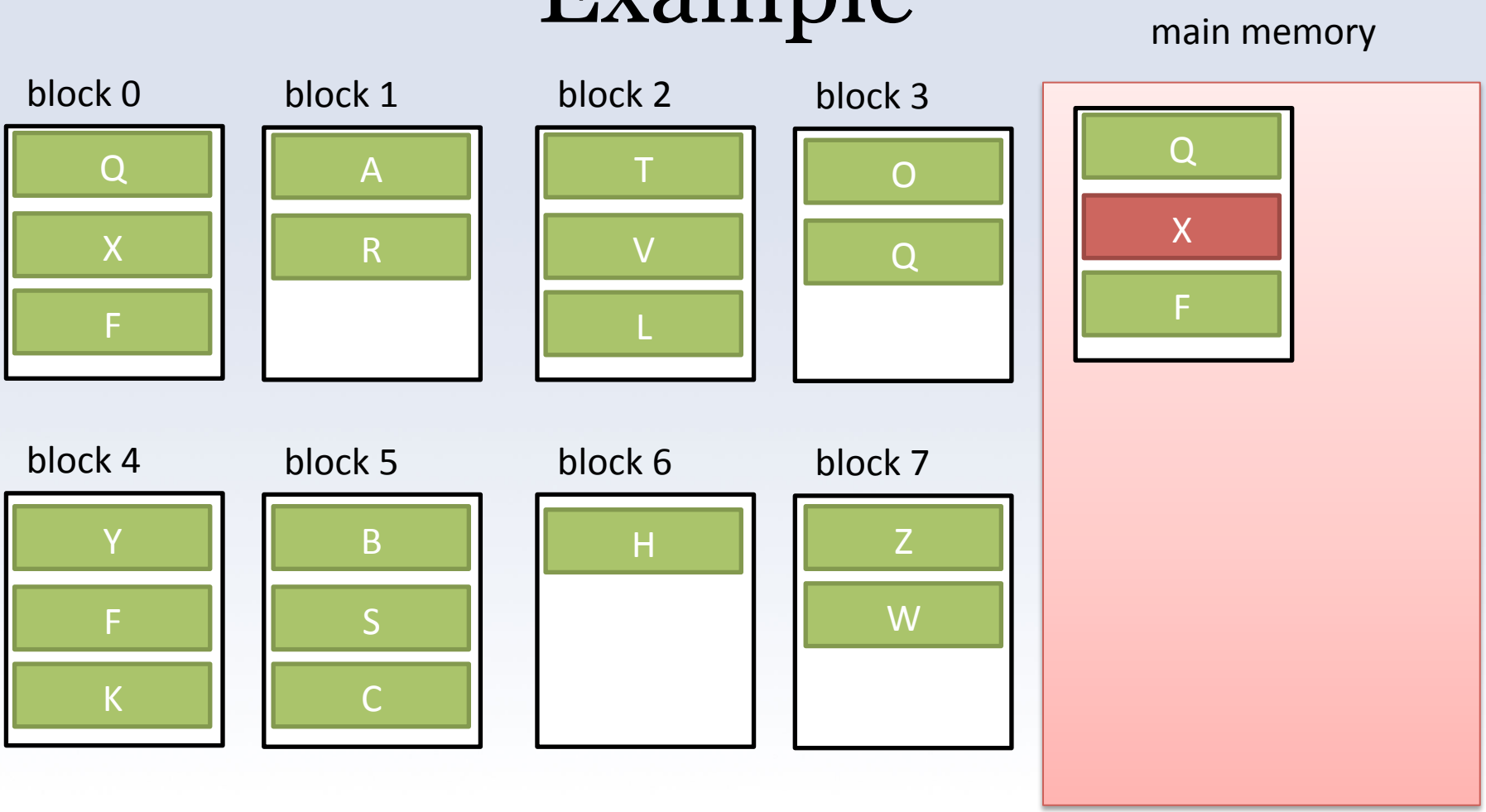
Example


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
block 6
H
block 7
Z
W
main memory
Q
X
F

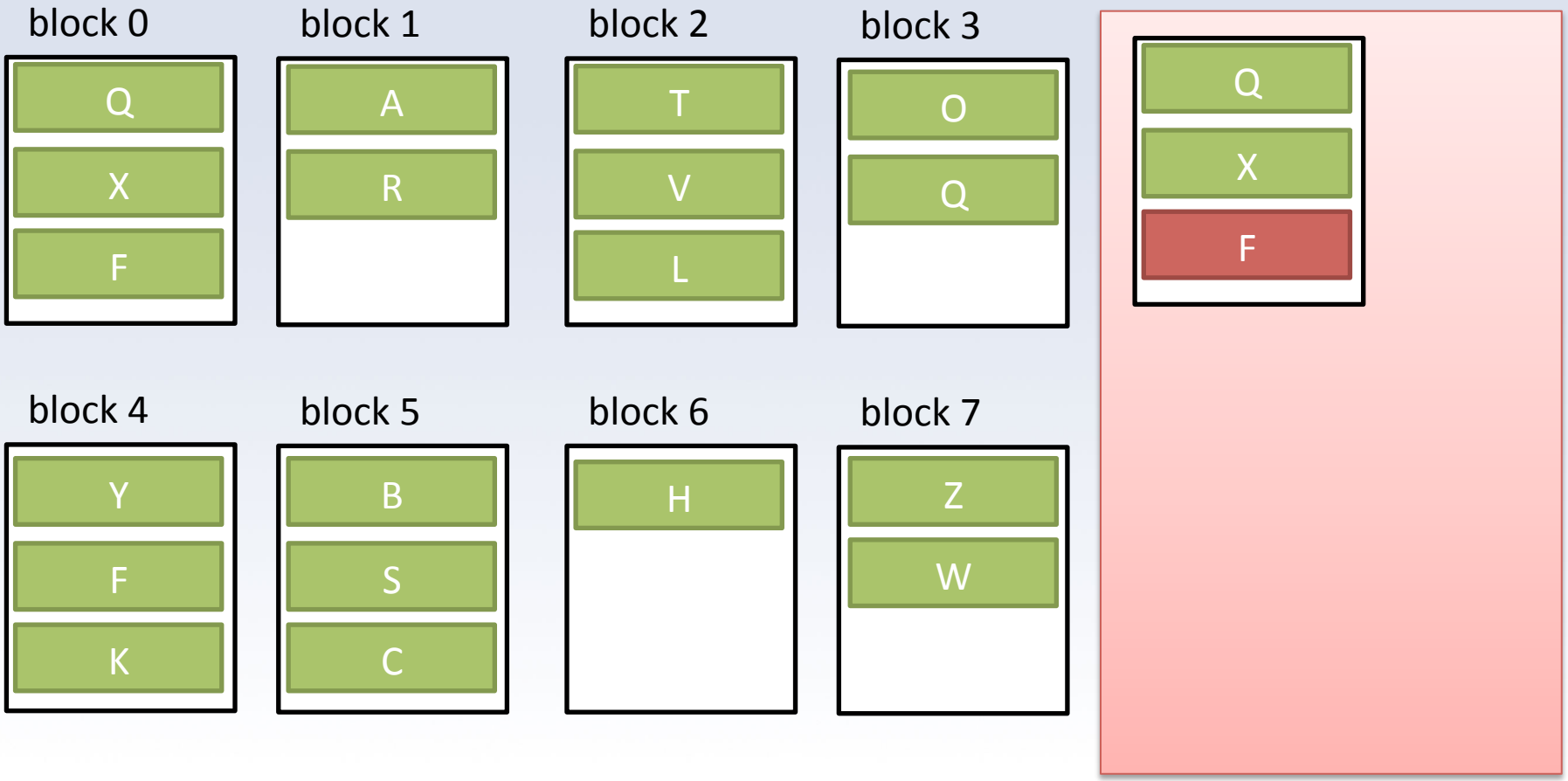
Example


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
block 6
H
block 7
Z
W
main memory
Q
X
F

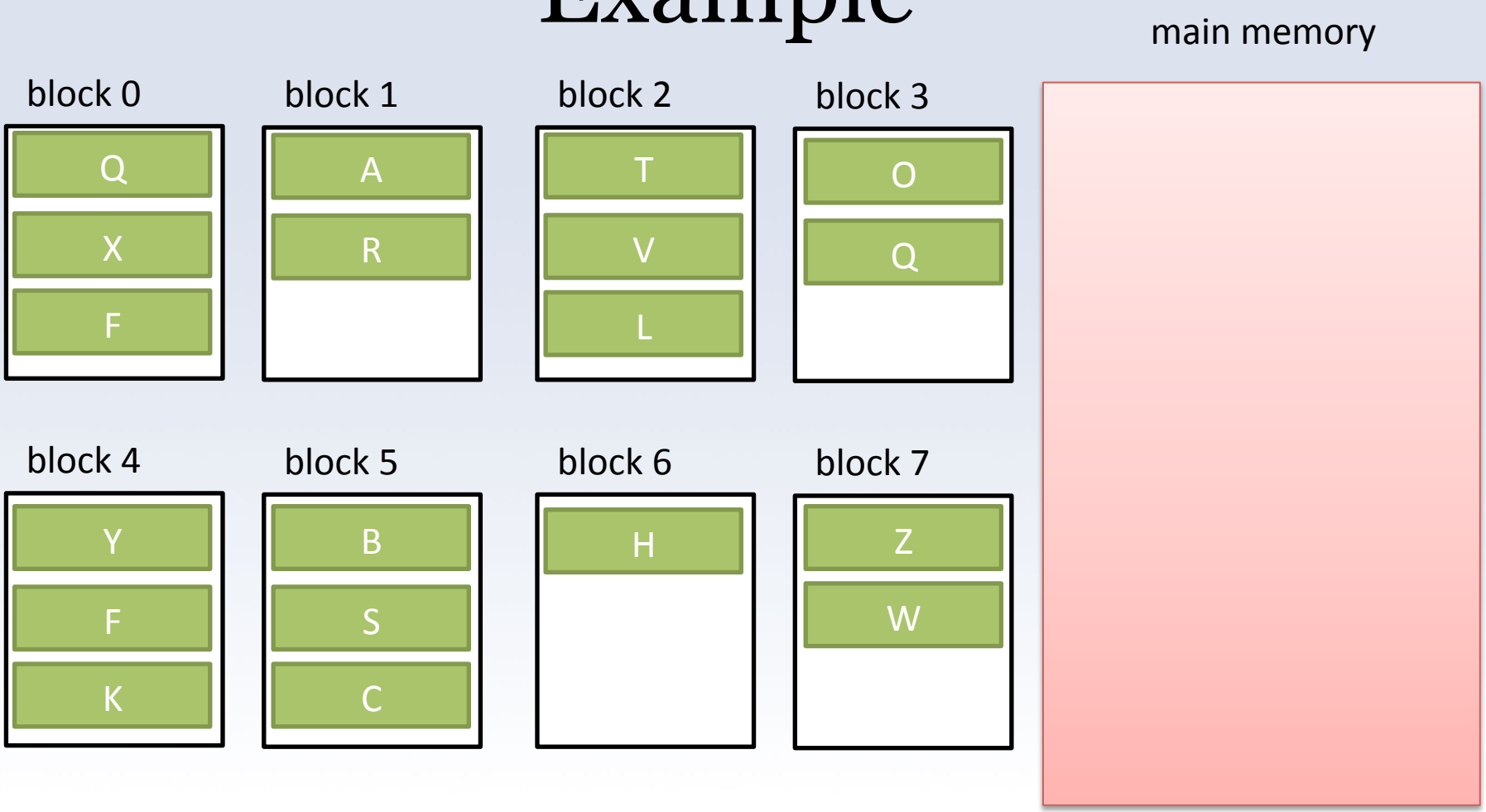
Example


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
block 6
H
block 7
Z
W
main memory
Q
X
F

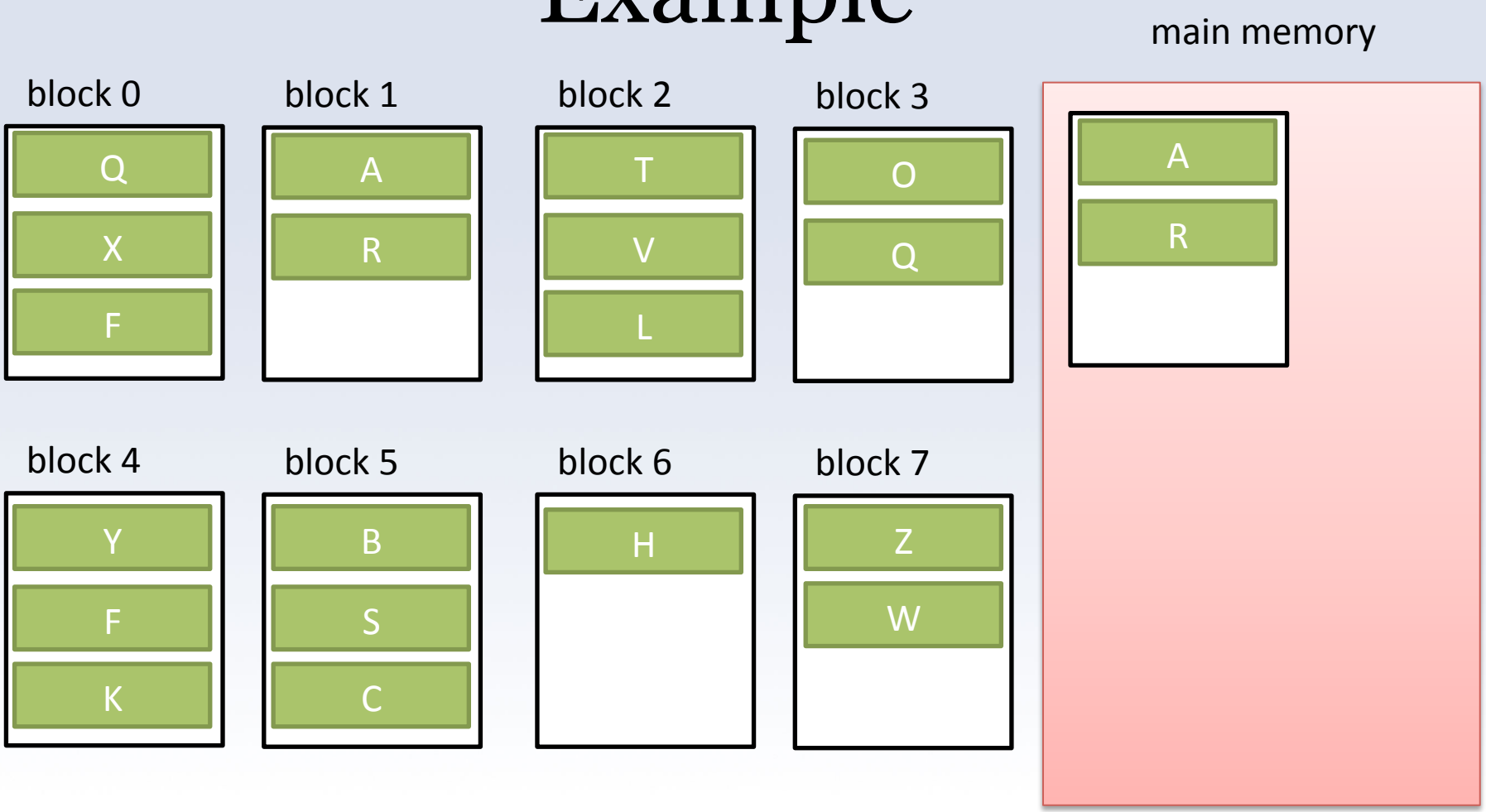
Example


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
block 6
H
block 7
Z
W
main memory
Q
X
F

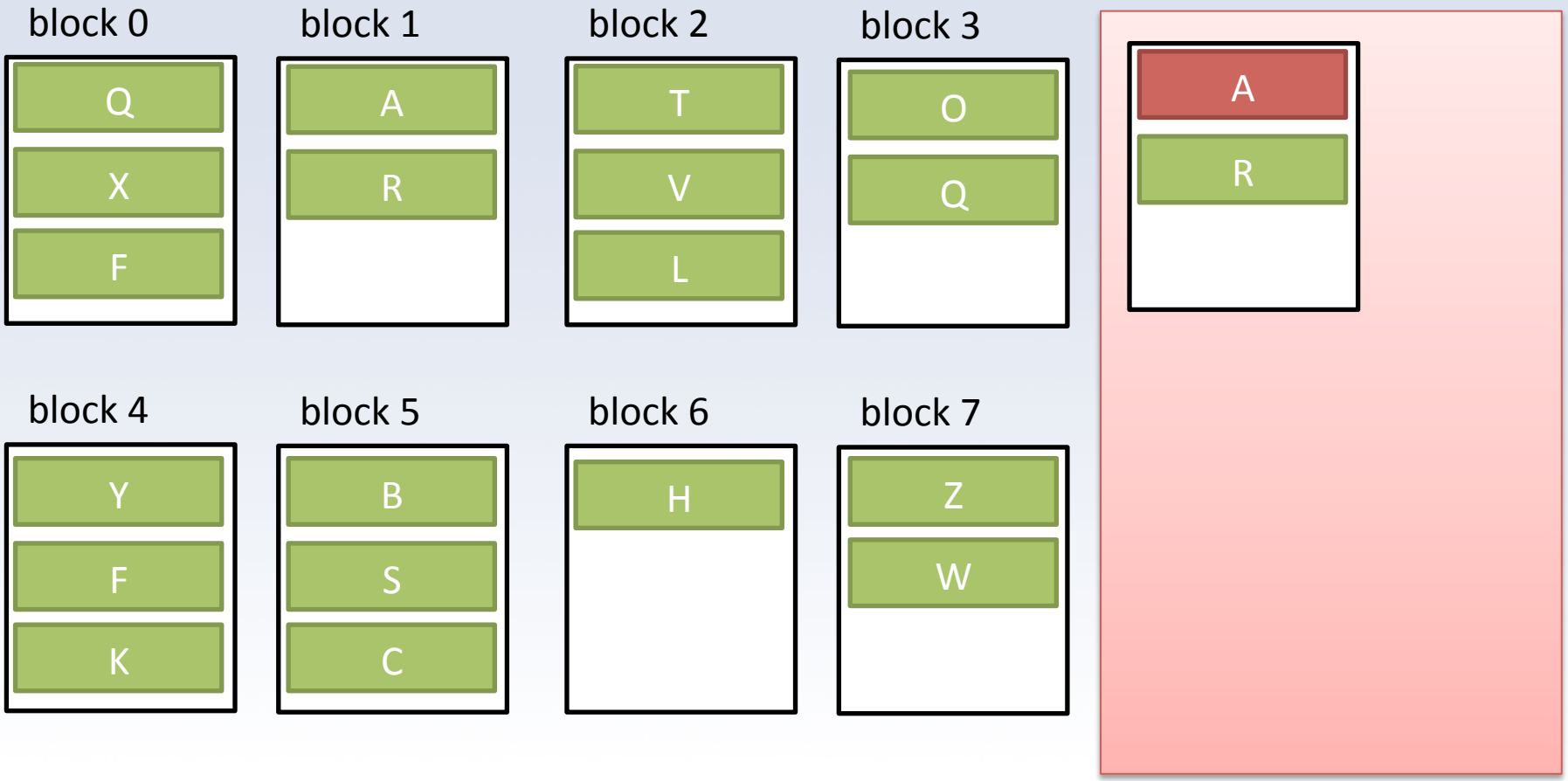
Example


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
block 6
H
block 7
Z
W
main memory

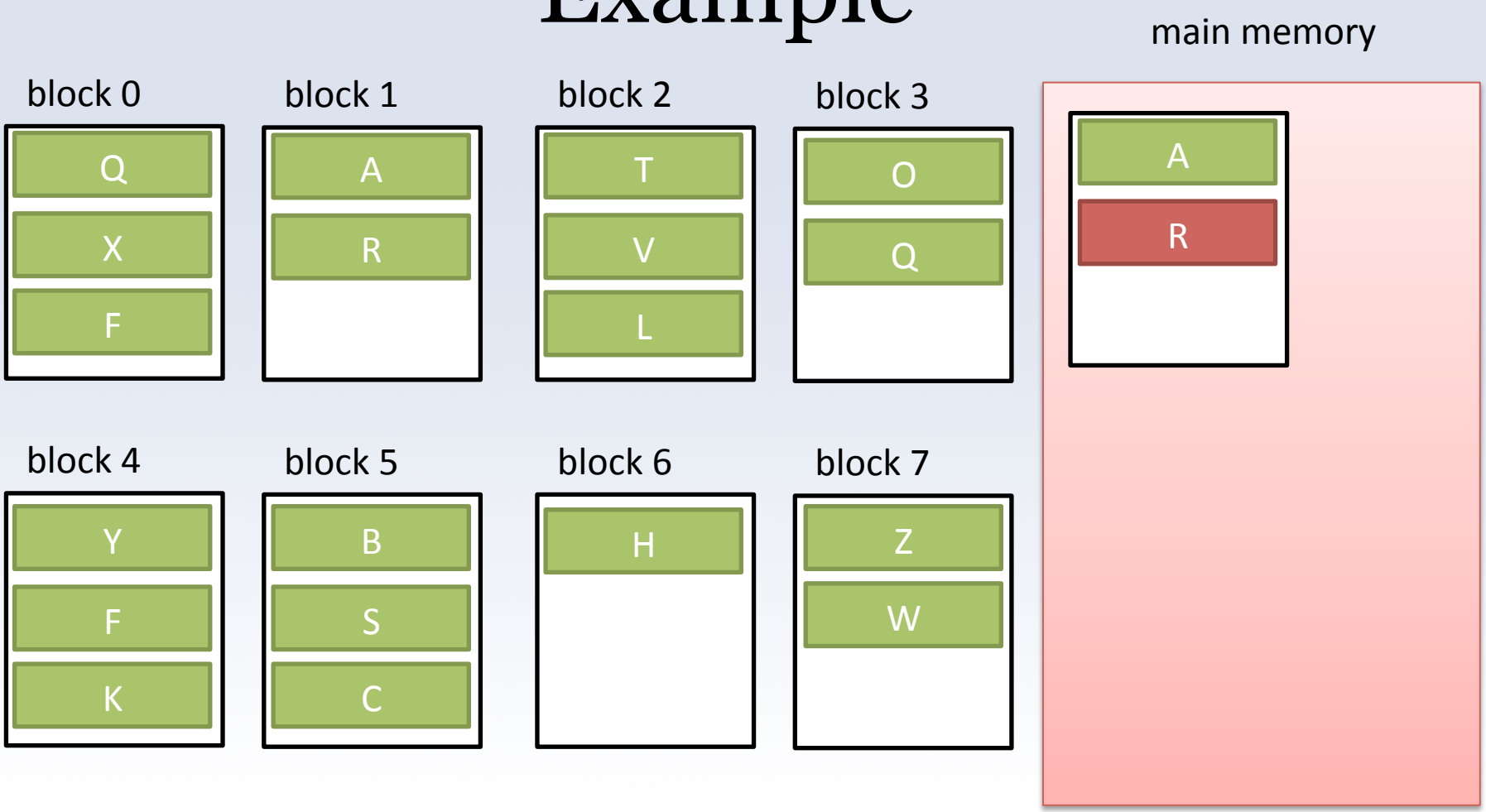
Example


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
block 6
H
block 7
Z
W
main memory
A
R

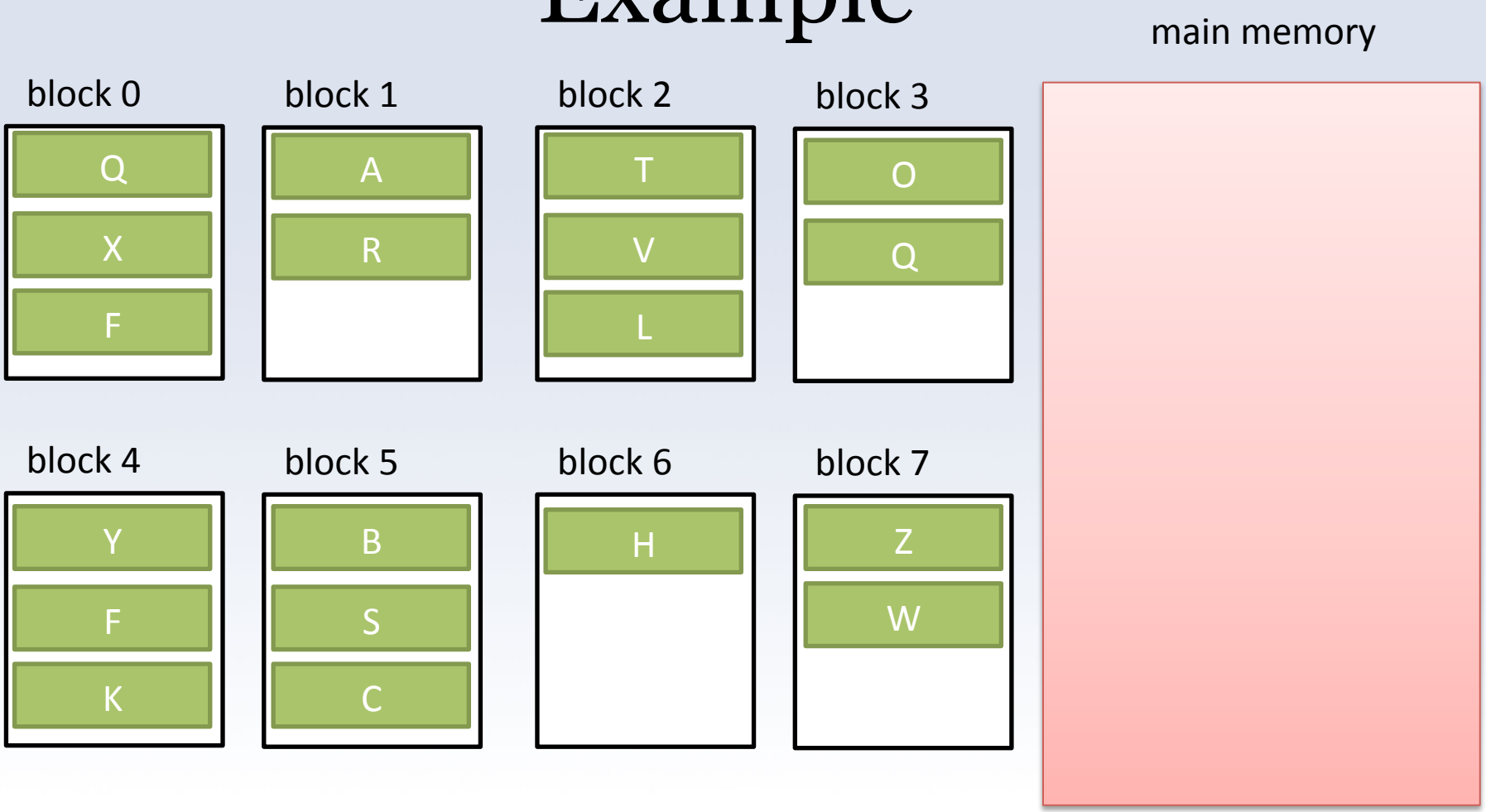
Example


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
block 6
H
block 7
Z
W
main memory
A
R

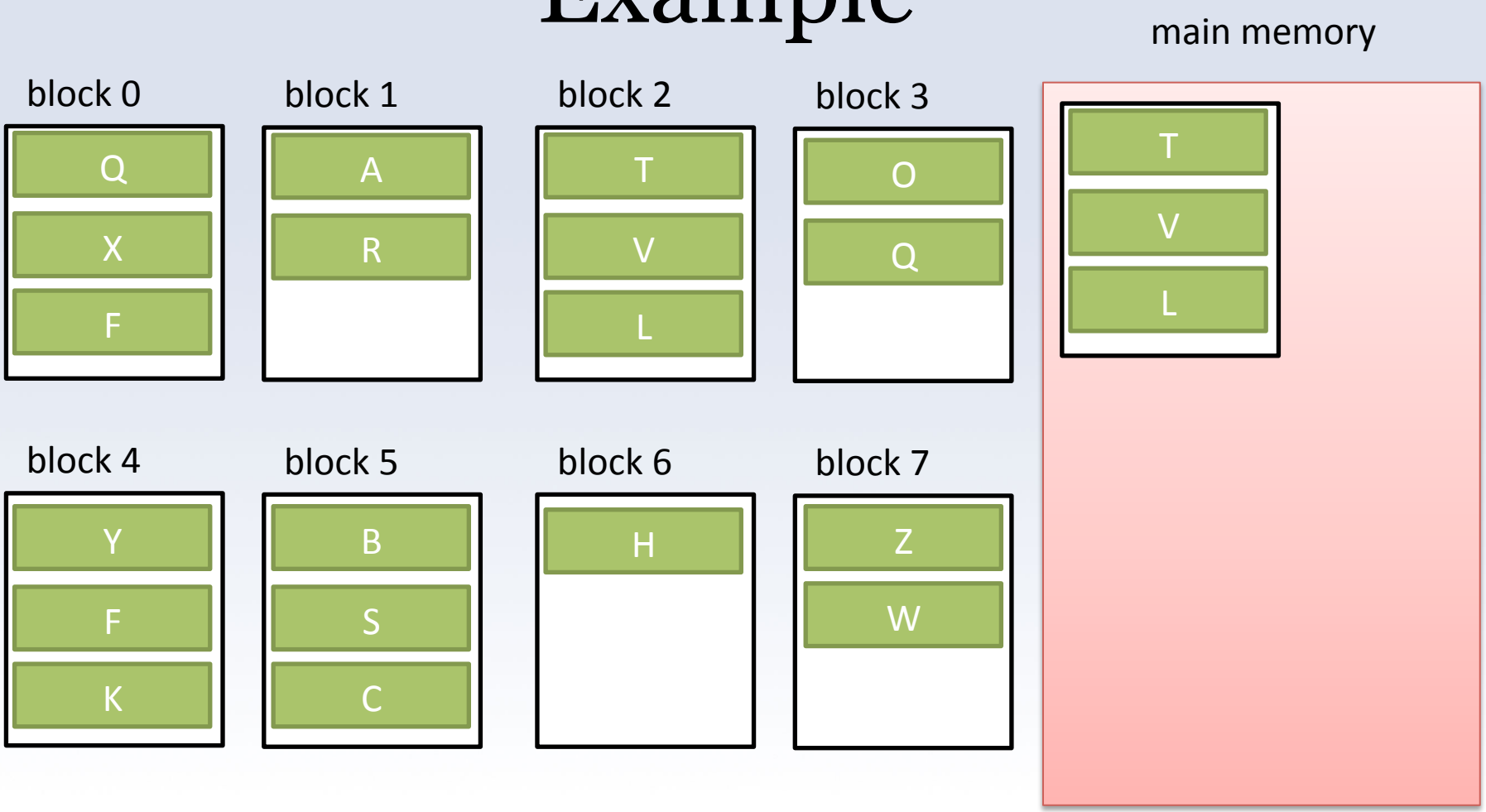
Example


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
block 6
H
block 7
Z
W
main memory
A
R

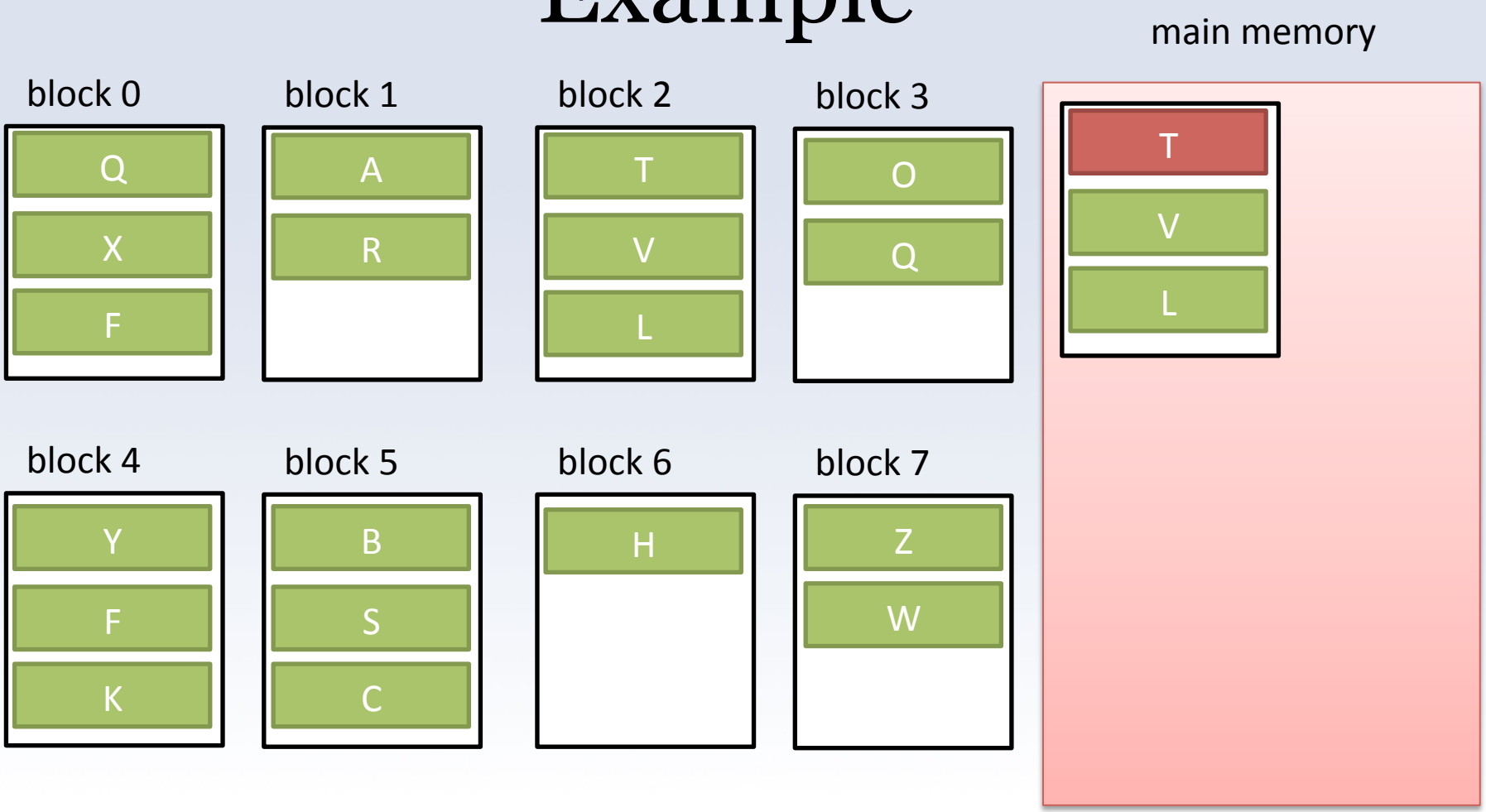
Example


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
block 6
H
block 7
Z
W
main memory

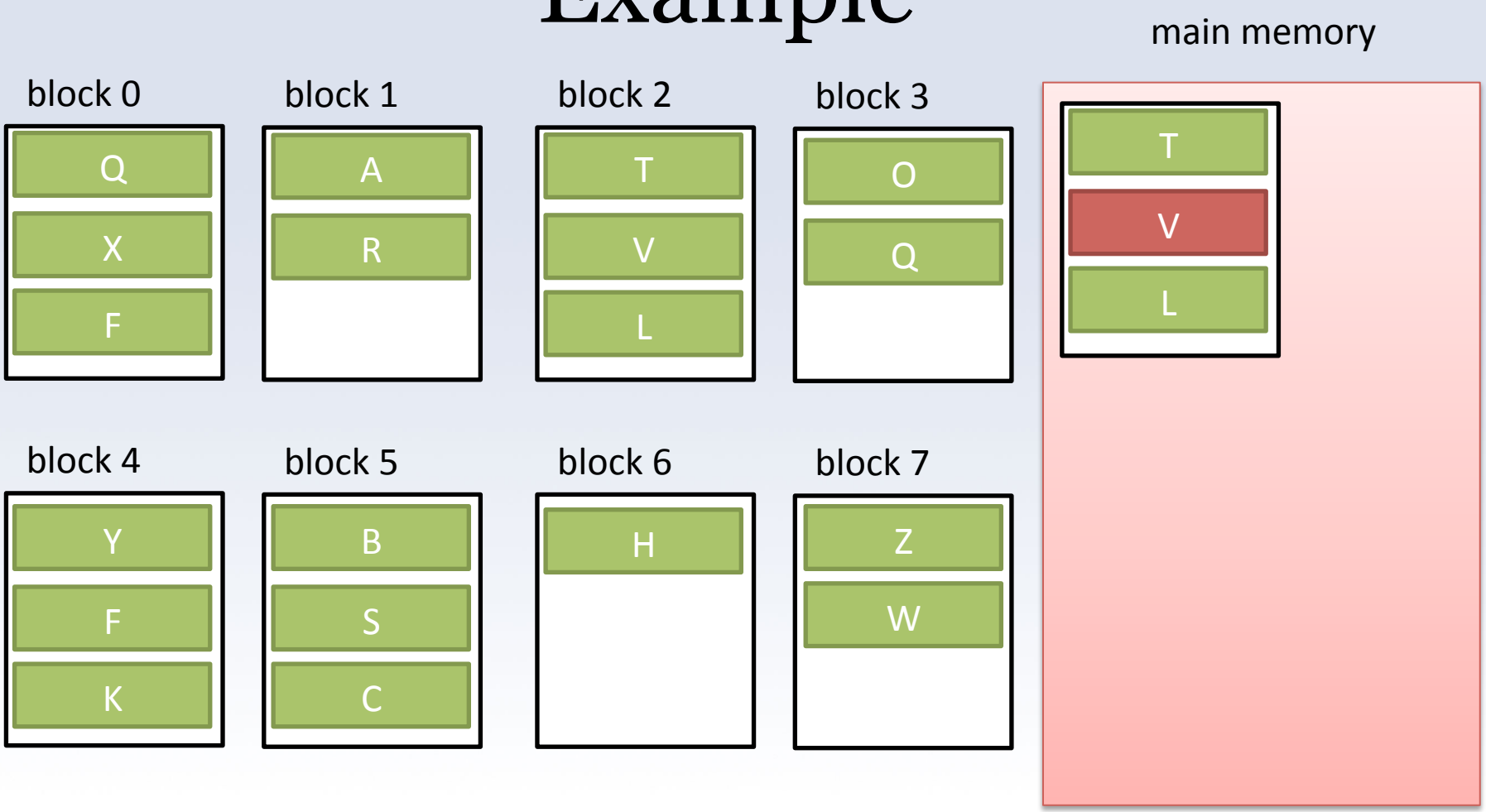
Example


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
block 6
H
block 7
Z
W
main memory
T
V
L

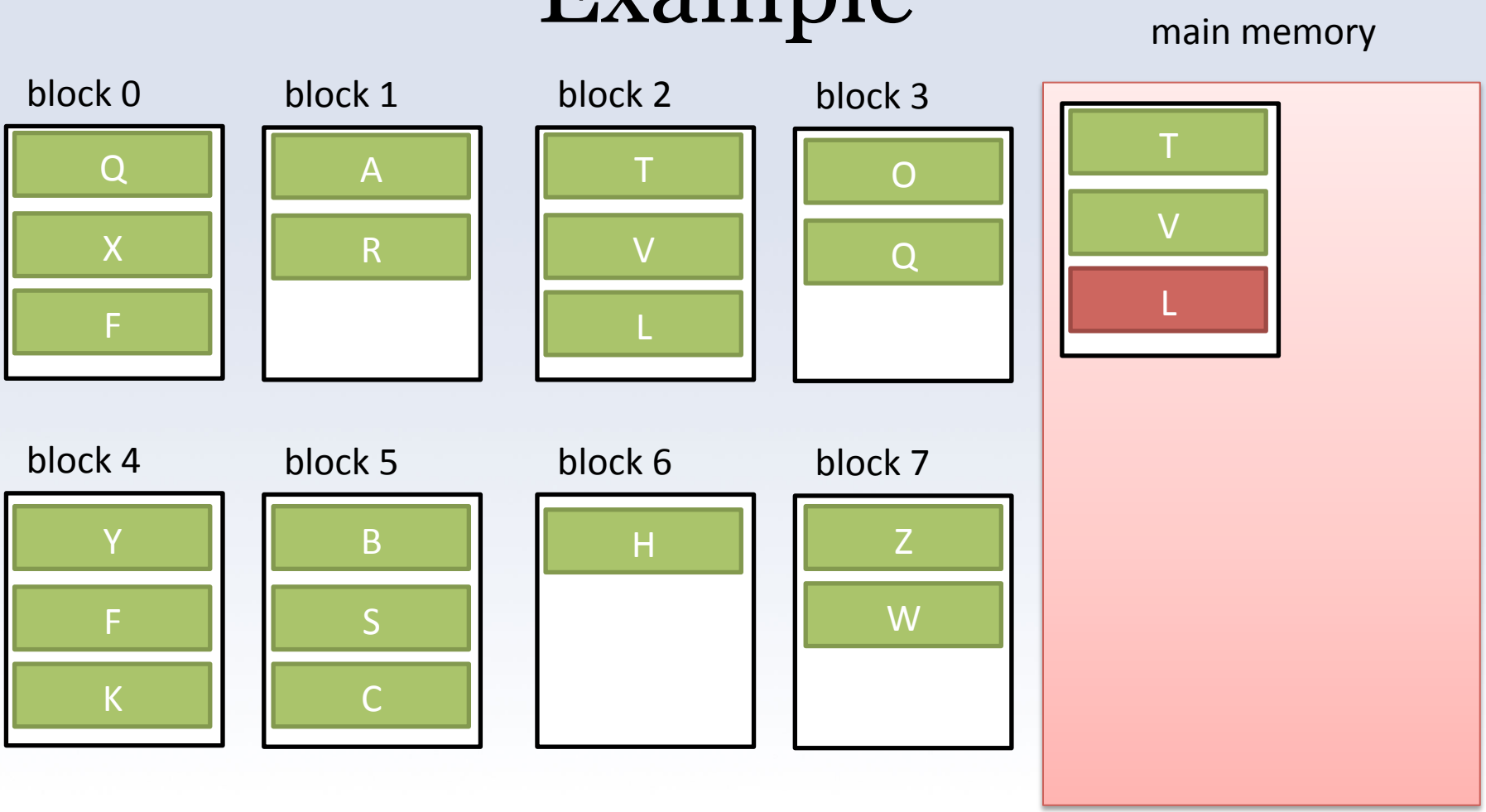
Example


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
block 6
H
block 7
Z
W
main memory
T
V
L

Example


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
block 6
H
block 7
Z
W
main memory
T
V
L

Example


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
block 6
H
block 7
Z
W
main memory
T
V
L

Cost of table scan

- Must read each block once
- Cost is therefore B(R)
  - number of blocks storing records from R

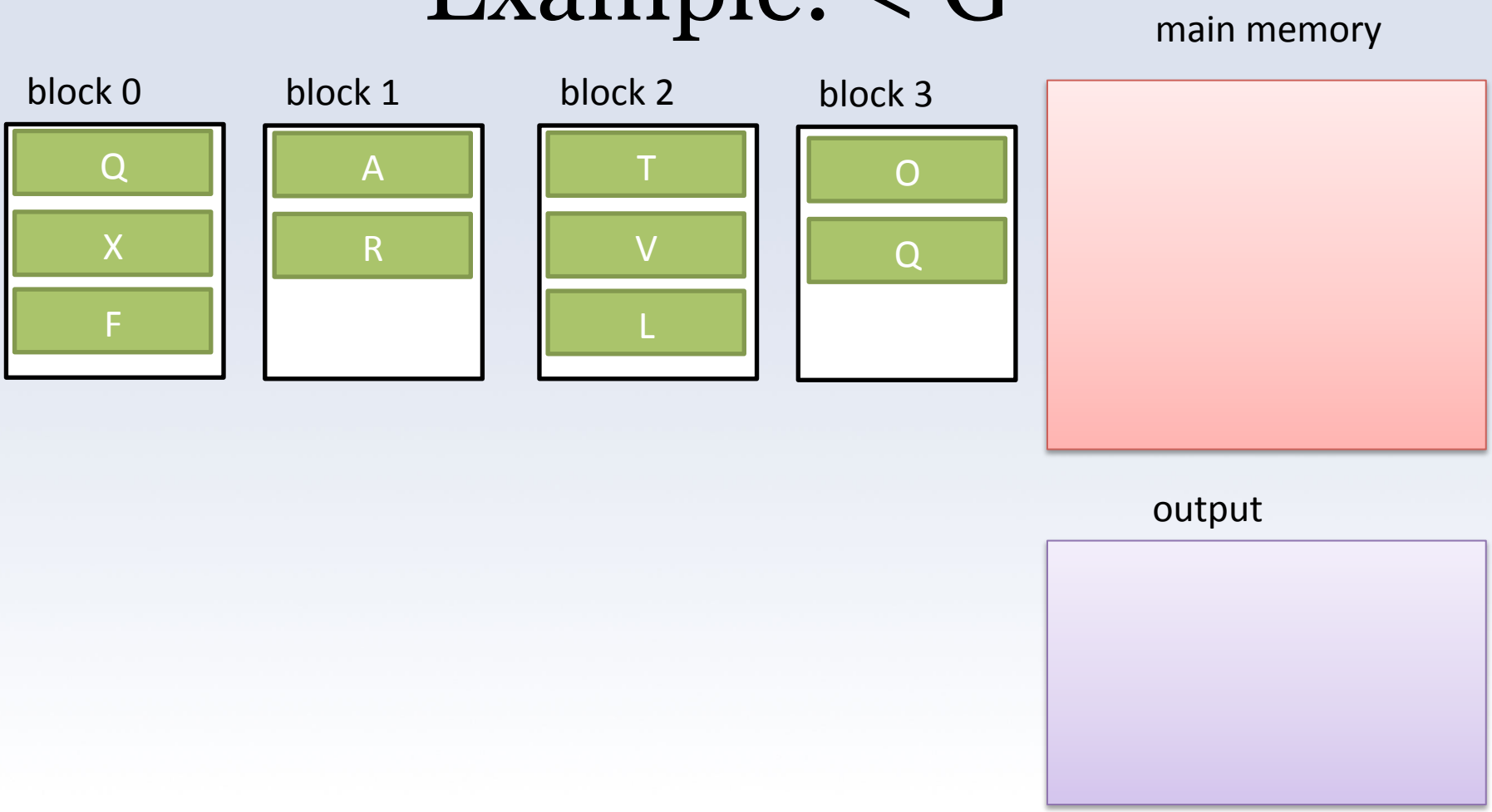
One-Pass Algorithms

- Sometimes, we can do an operation passing over the data once
  - Example: table scan
- Usually, relation has to be small
- You probably implemented some of these in MP1

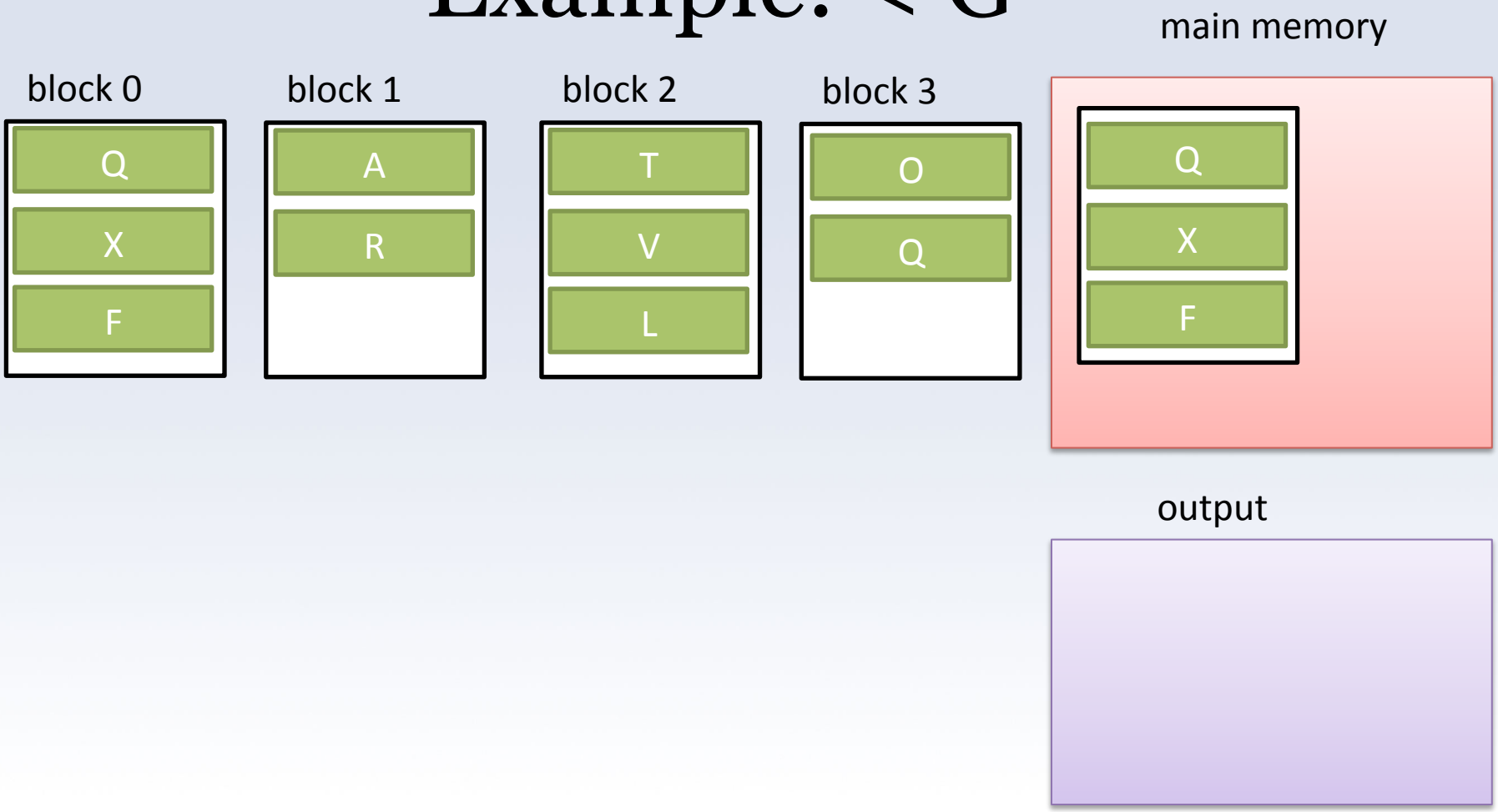
Tuple-at-a-Time

- σ and π require *at most* one pass:
  - do a table scan, but only return desired rows/ columns
- Cost: B(R)

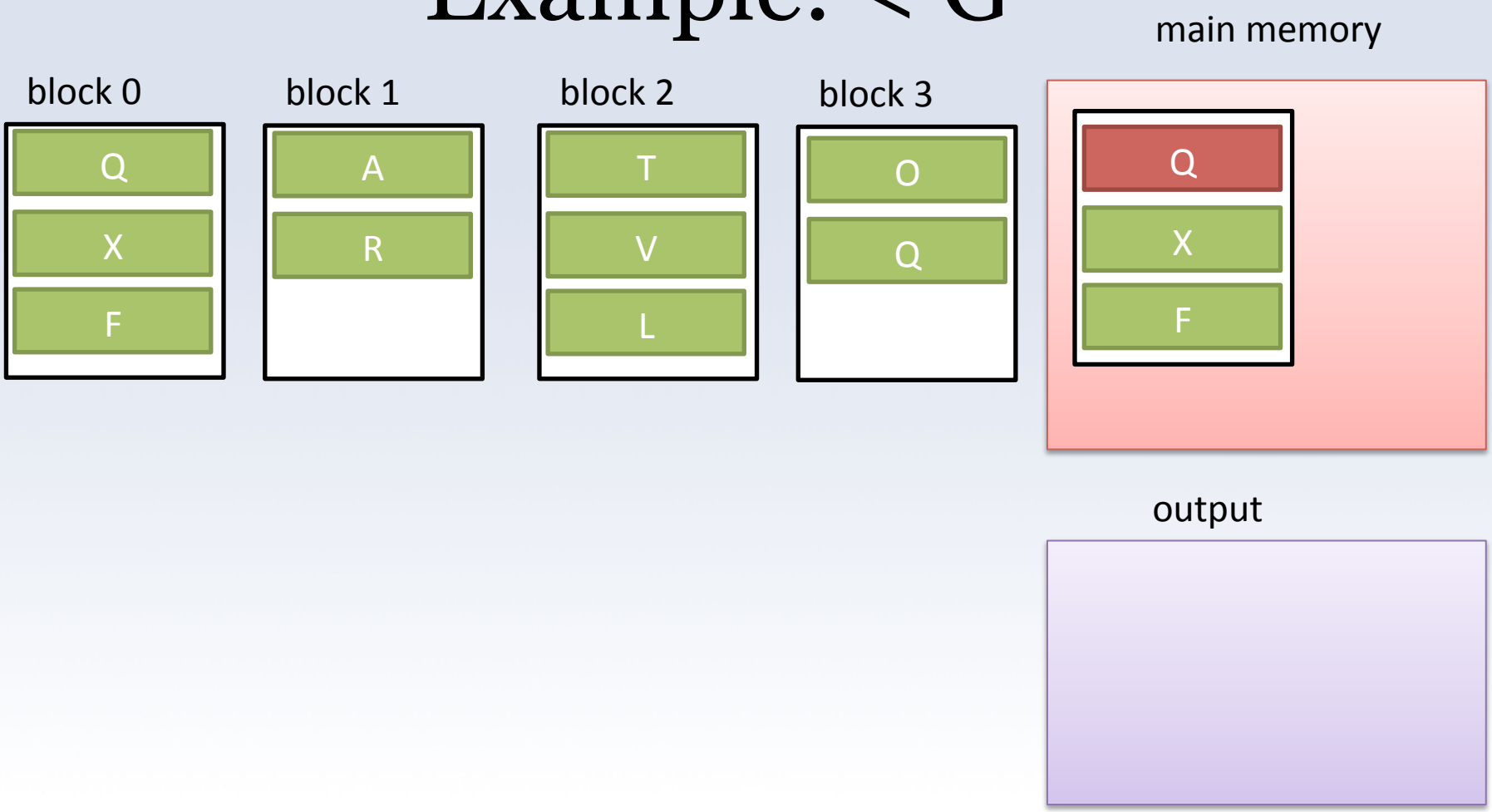
Example: <'G'


block 0
Q
X
F
block 1
A
R
block 2
T
V
L
block 3
O
Q
main memory
output

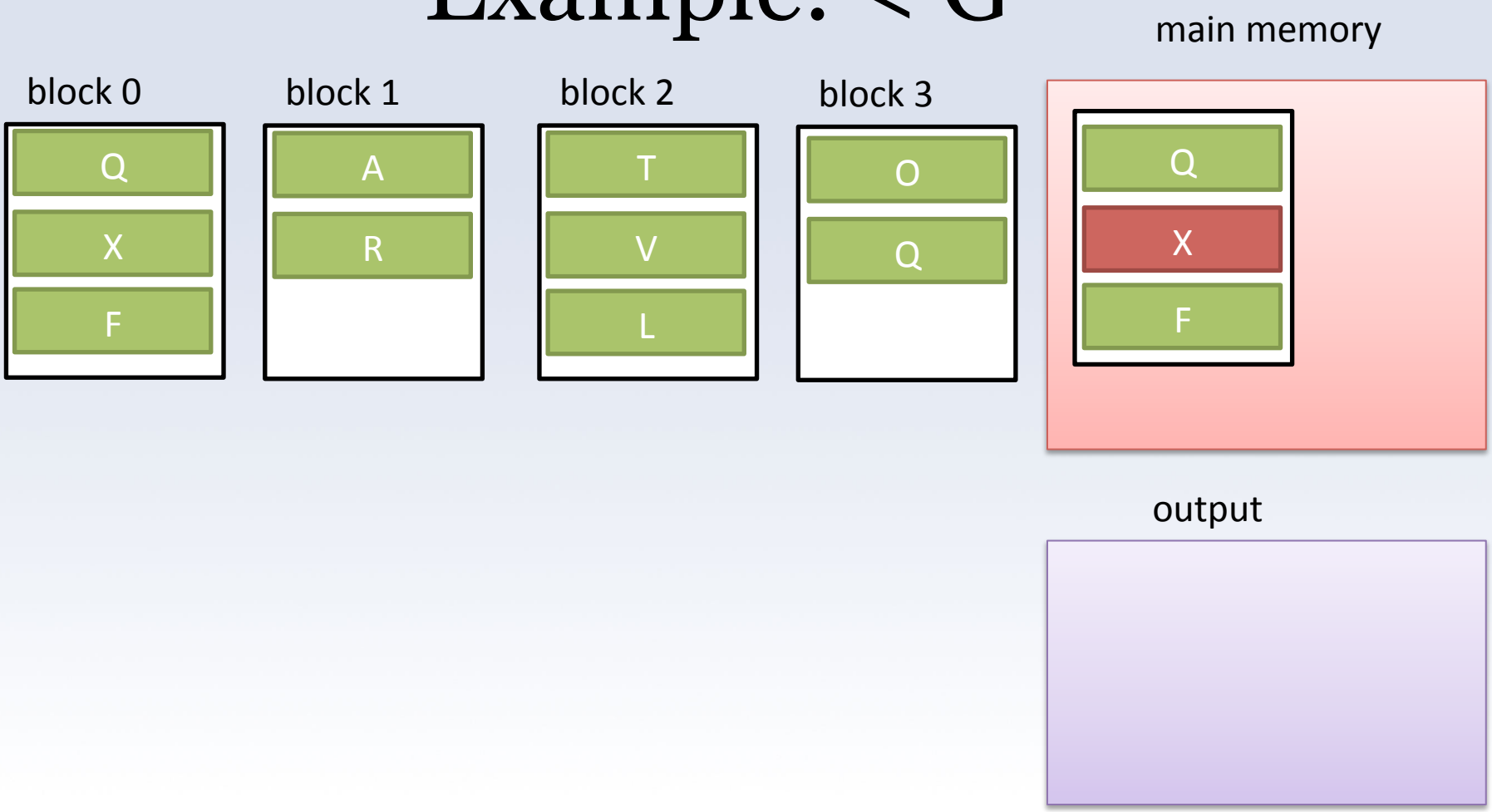
Example: <'G'


block 0
Q
X
F
block 1
A
R
block 2
T
V
L
block 3
O
Q
main memory
Q
X
F
output

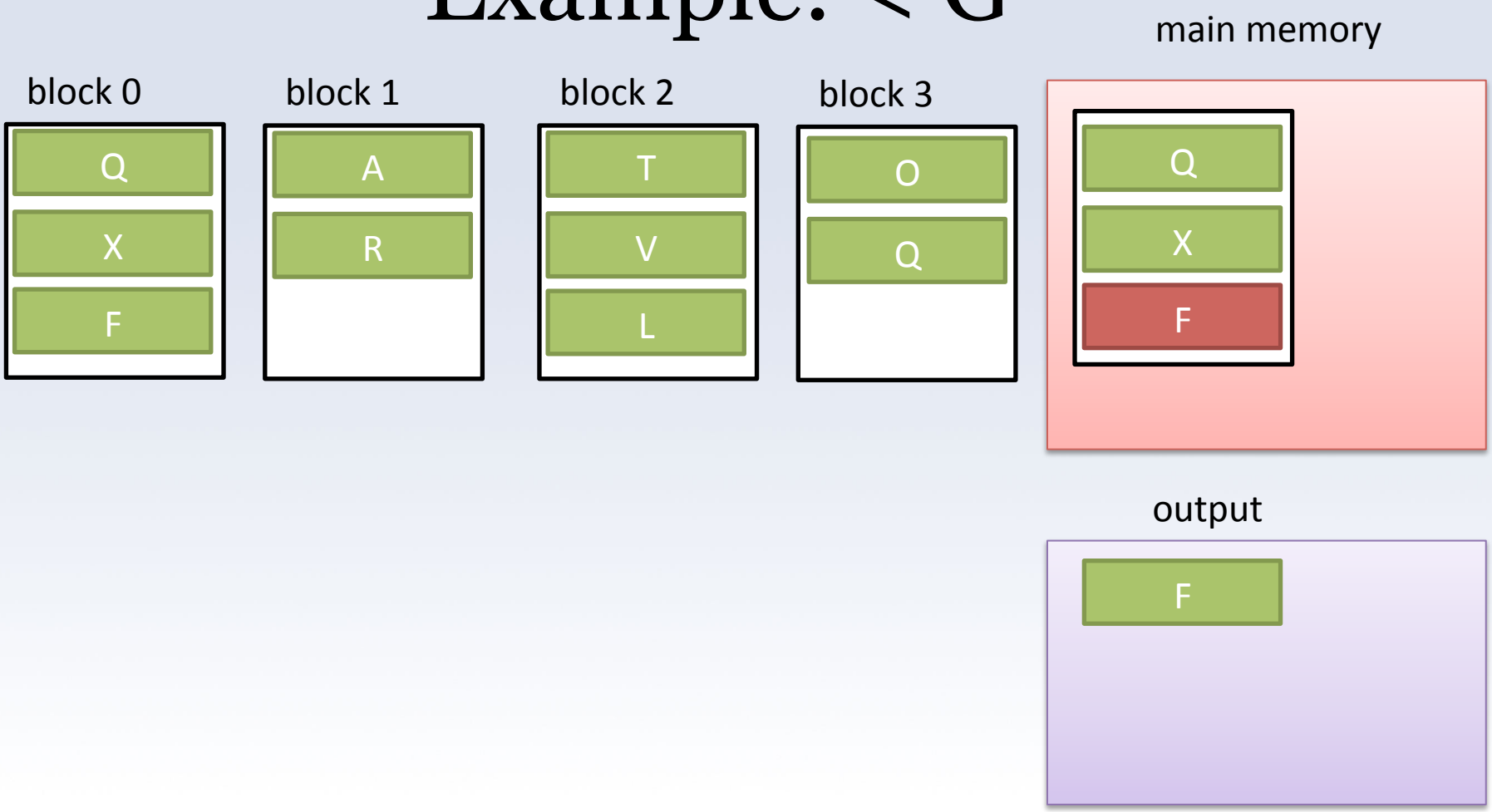
Example: <'G'


block 0
Q
X
F
block 1
A
R
block 2
T
V
L
block 3
O
Q
main memory
Q
X
F
output

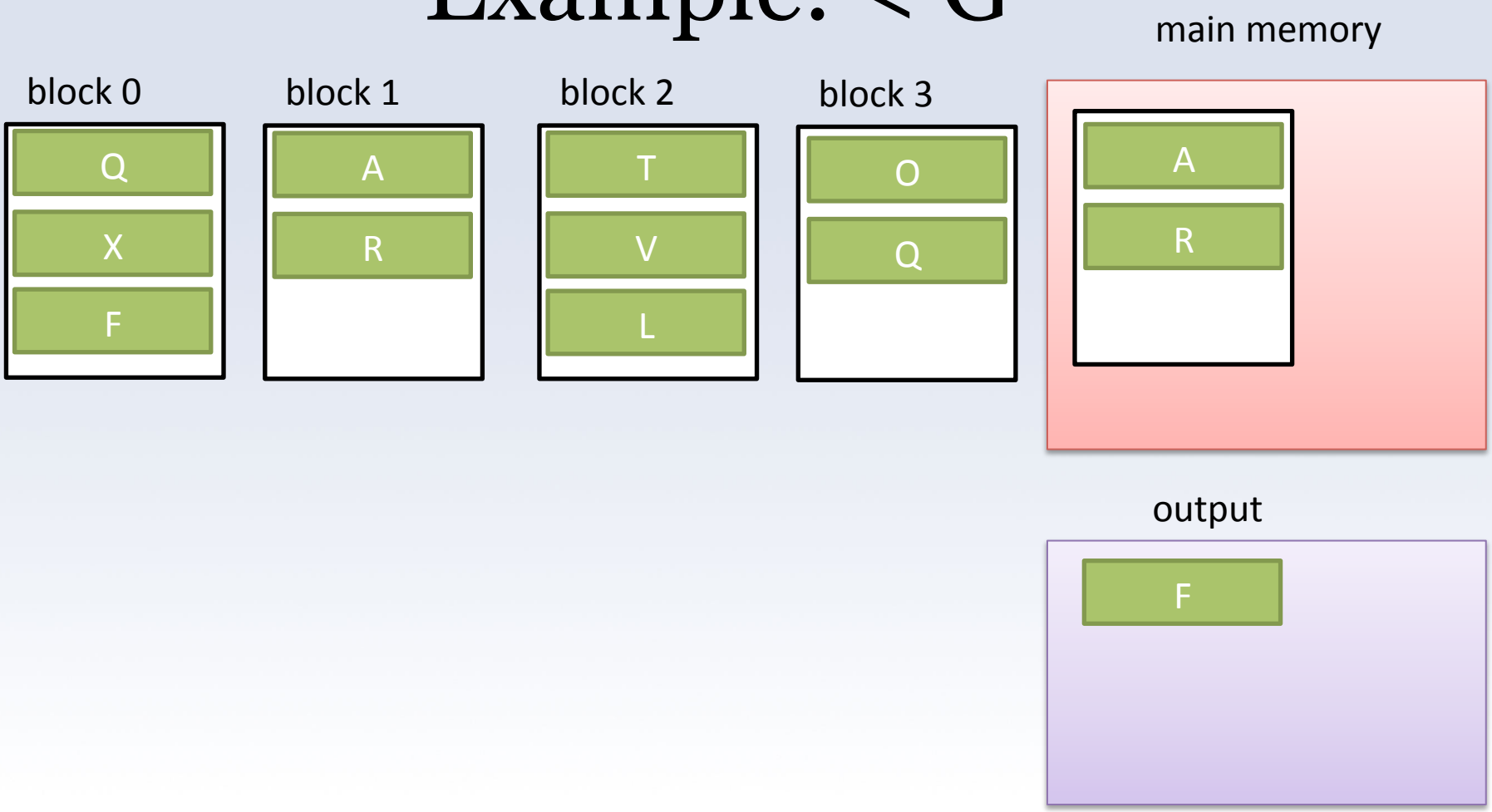
Example: <'G'


block 0
Q
X
F
block 1
A
R
block 2
T
V
L
block 3
O
Q
main memory
Q
X
F
output

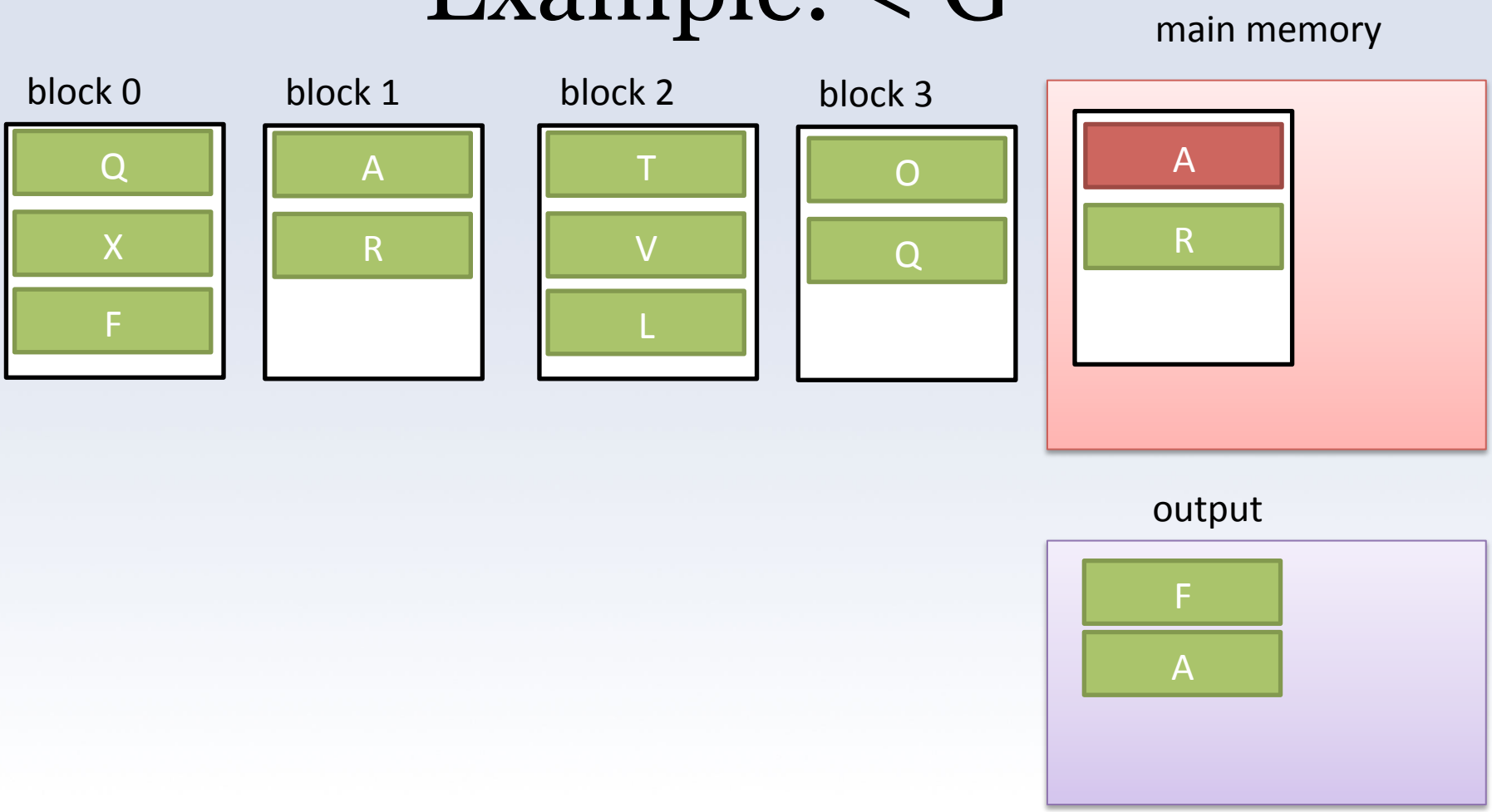
Example: <'G'


block 0
Q
X
F
block 1
A
R
block 2
T
V
L
block 3
O
Q
main memory
Q
X
F
output
F

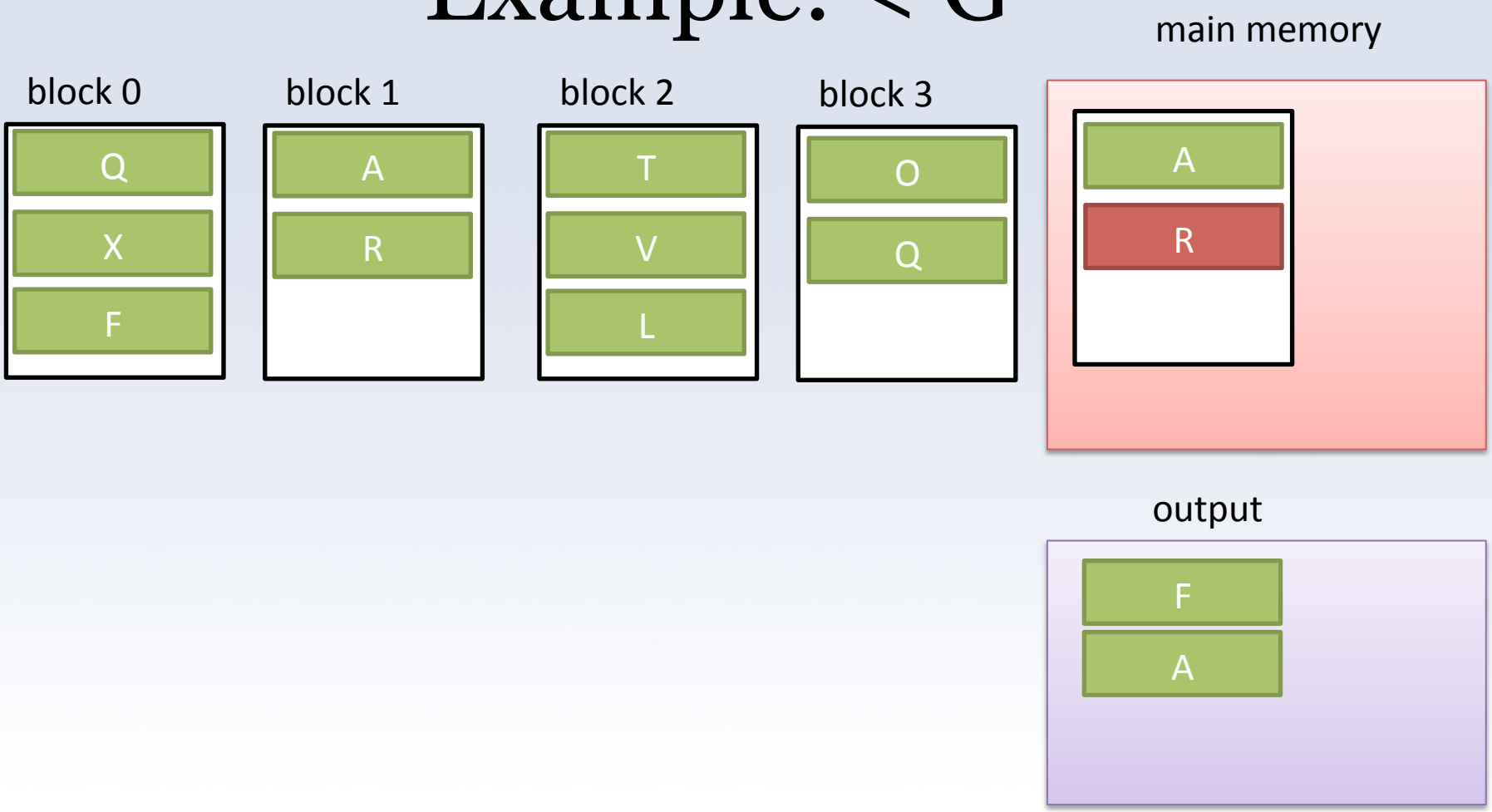
Example: <'G'


block 0
Q
X
F
block 1
A
R
block 2
T
V
L
block 3
O
Q
main memory
A
R
output
F

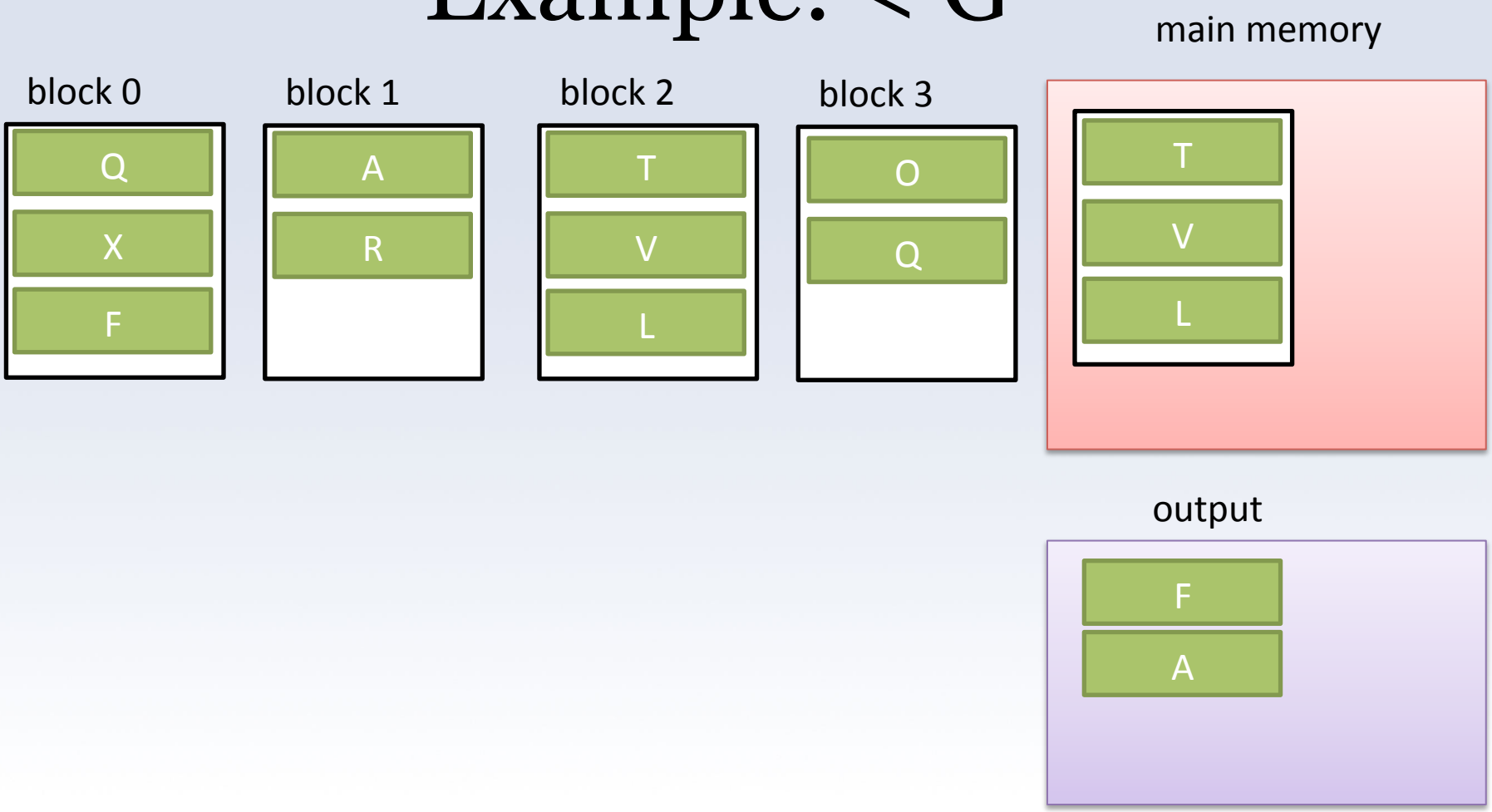
Example: <'G'


block 0
Q
X
F
block 1
A
R
block 2
T
V
L
block 3
O
Q
main memory
A
R
output
F
A

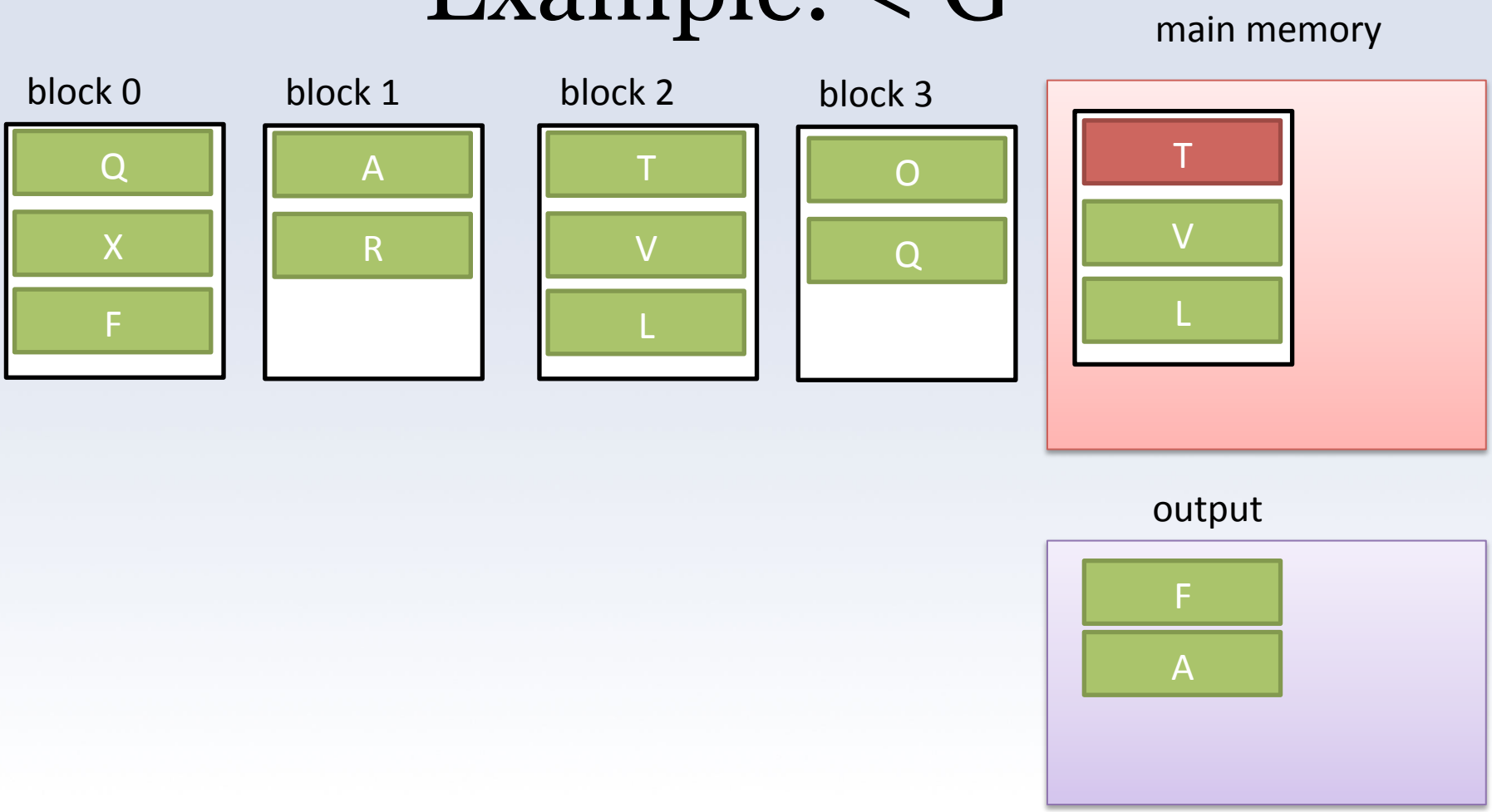
Example: <'G'


block 0
Q
X
F
block 1
A
R
block 2
T
V
L
block 3
O
Q
main memory
A
R
output
F
A

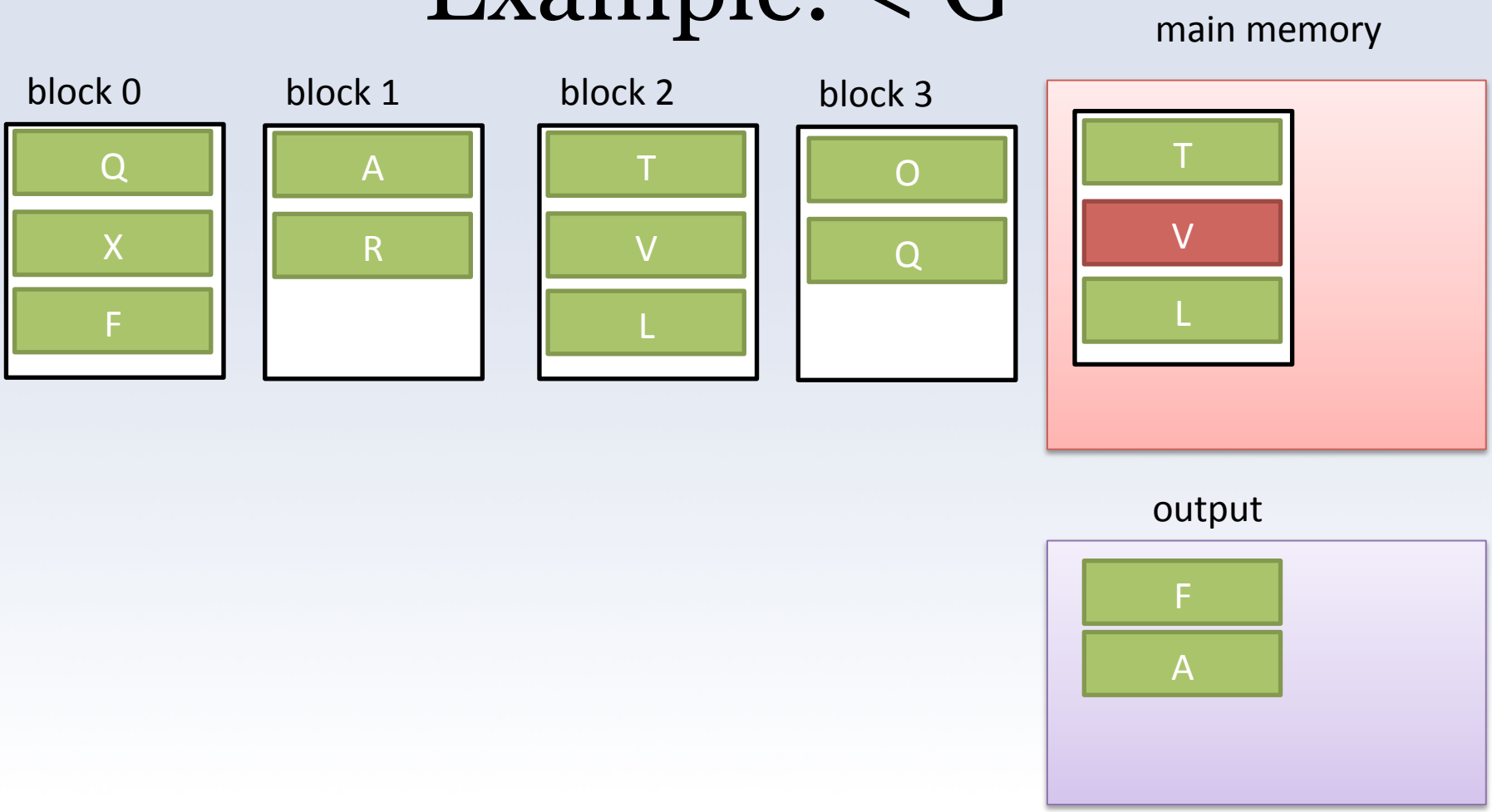
Example: <'G'


block 0
Q
X
F
block 1
A
R
block 2
T
V
L
block 3
O
Q
main memory
T
V
L
output
F
A

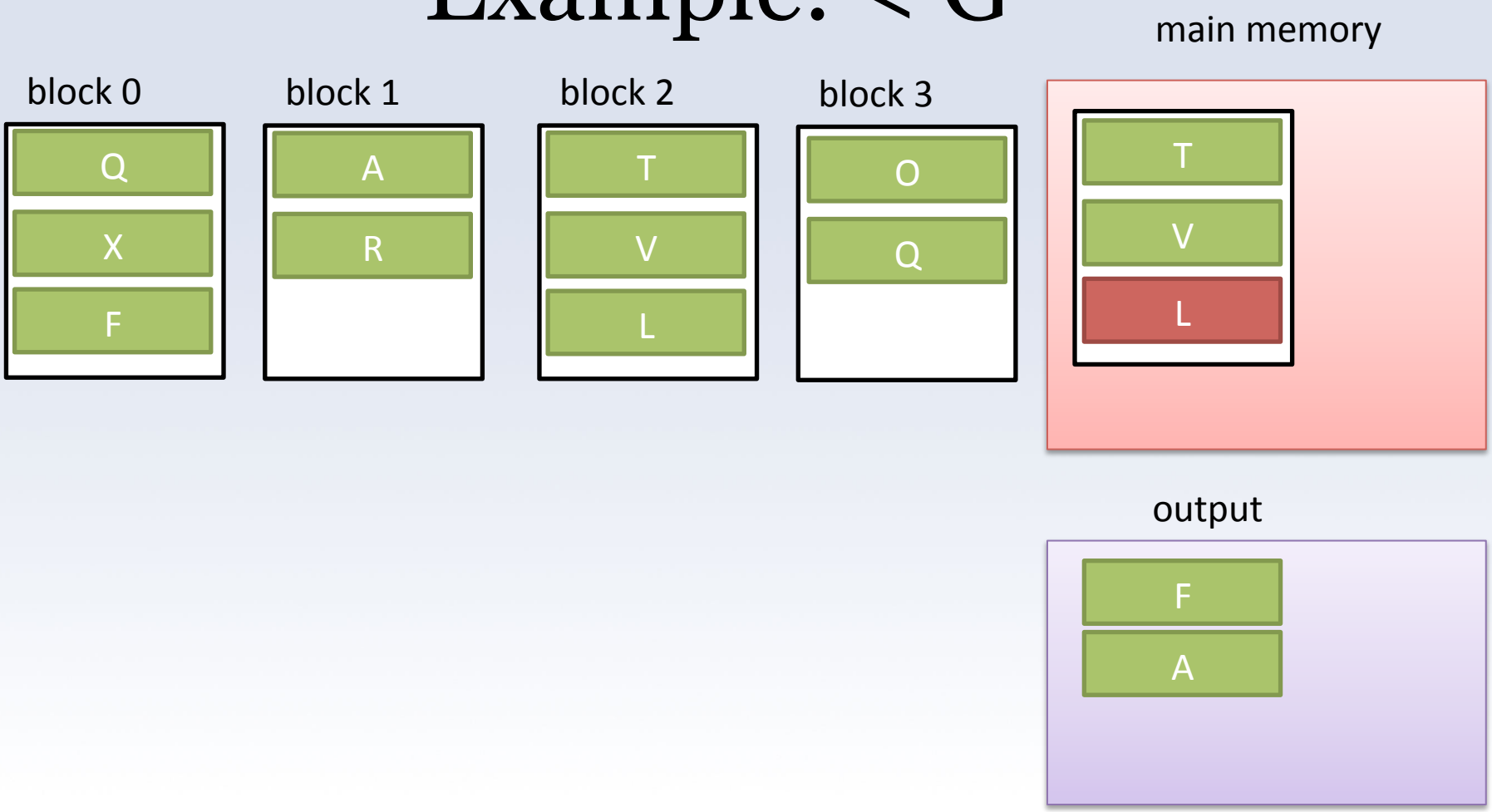
Example: <'G'


block 0
Q
X
F
block 1
A
R
block 2
T
V
L
block 3
O
Q
main memory
T
V
L
output
F
A

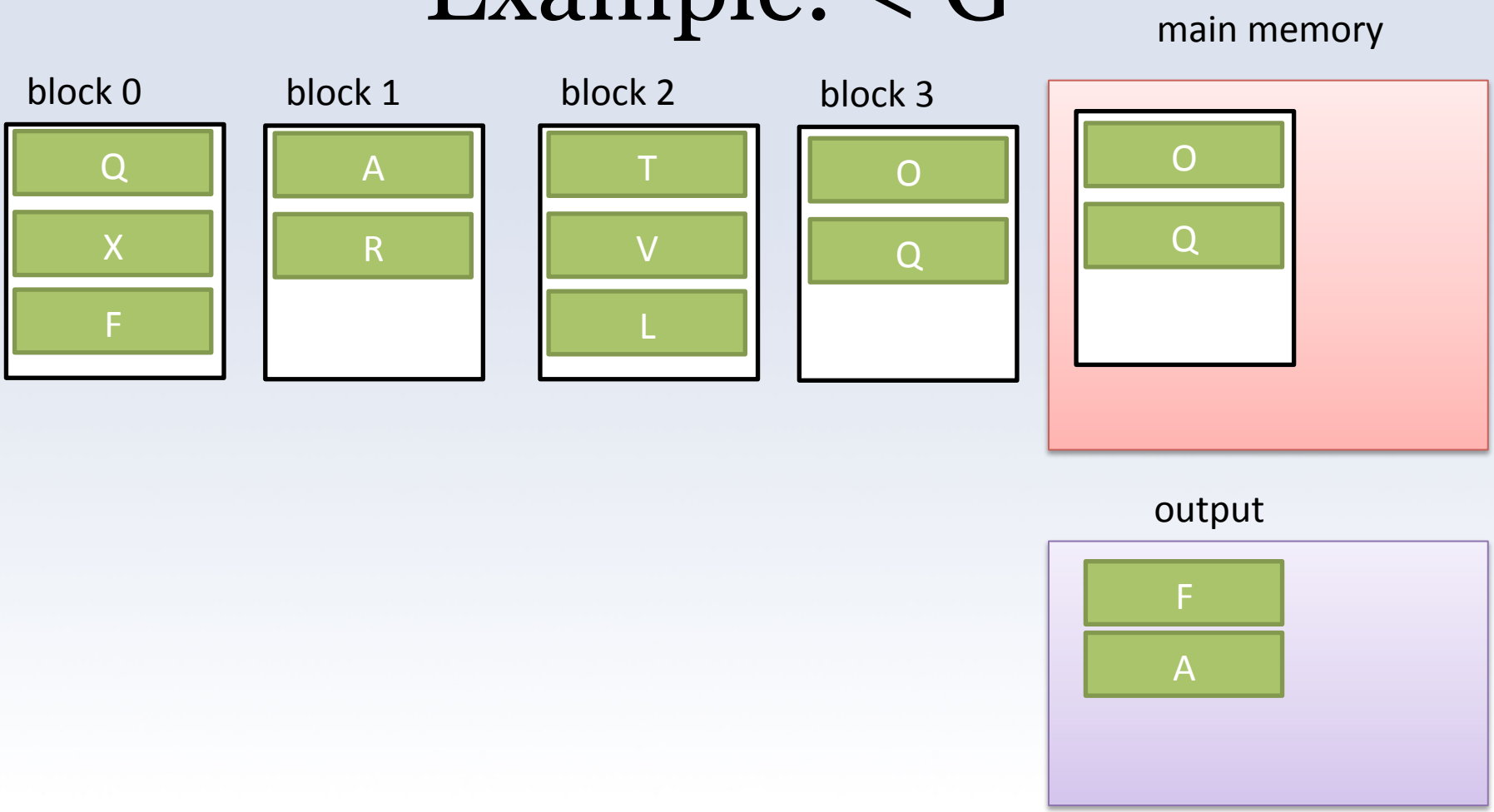
Example: <'G'


block 0
Q
X
F
block 1
A
R
block 2
T
V
L
block 3
O
Q
main memory
T
V
L
output
F
A

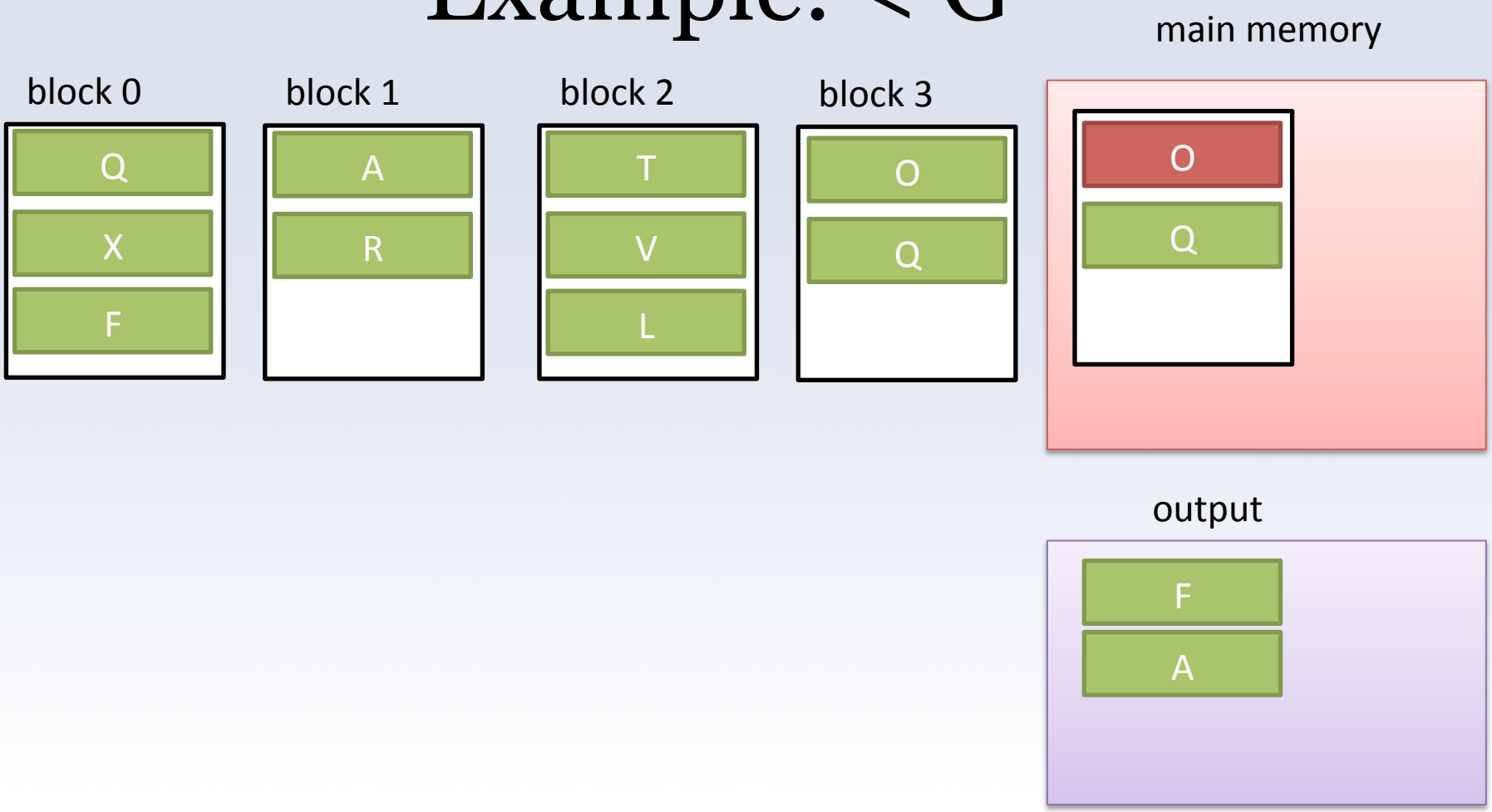
Example: <'G'


block 0
Q
X
F
block 1
A
R
block 2
T
V
L
block 3
O
Q
main memory
T
V
L
output
F
A

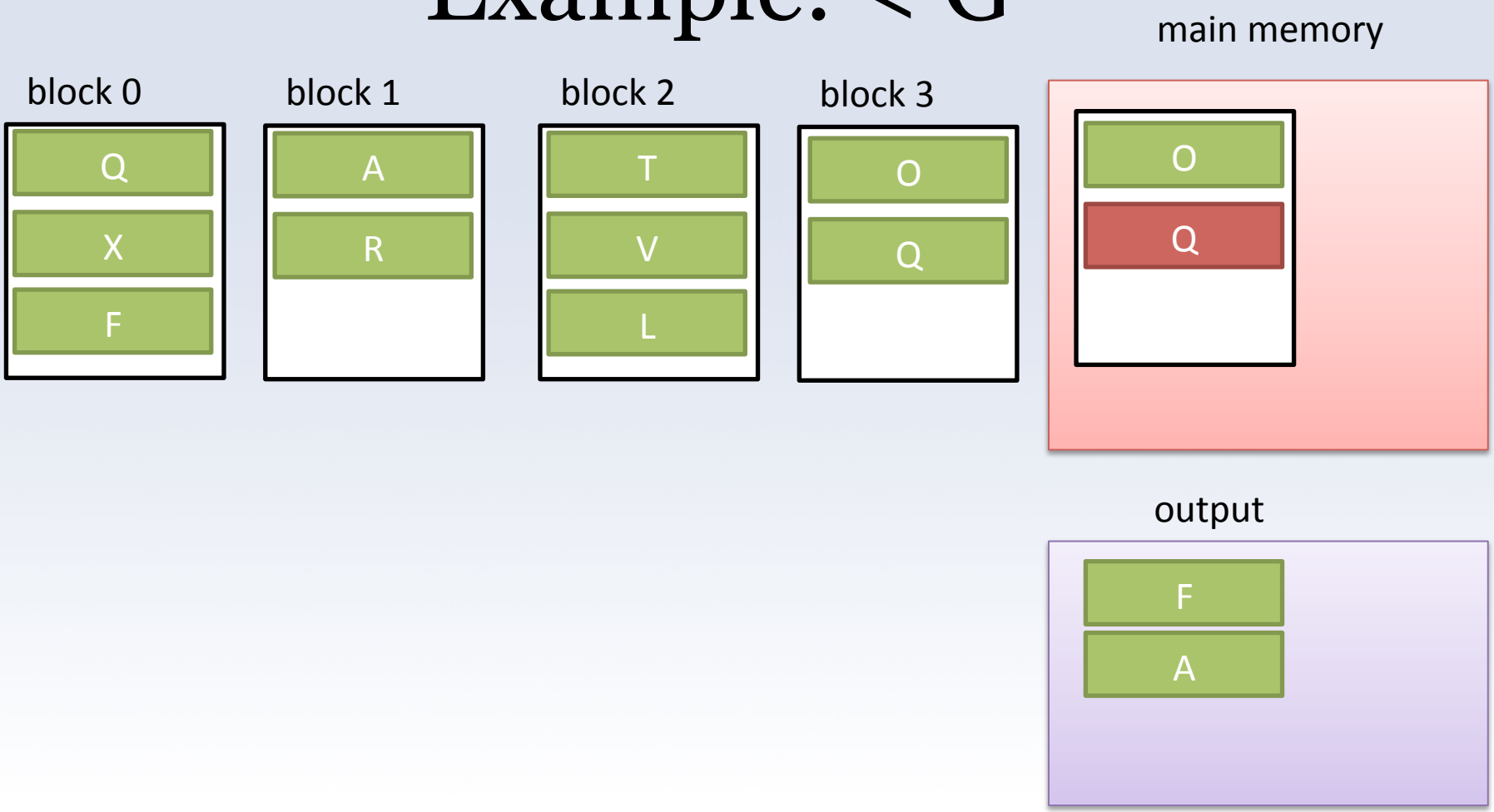
Example: <'G'


block 0
Q
X
F
block 1
A
R
block 2
T
V
L
block 3
O
Q
main memory
O
Q
output
F
A

Example: <'G'


block 0
Q
X
F
block 1
A
R
block 2
T
V
L
block 3
O
Q
main memory
O
Q
output
F
A

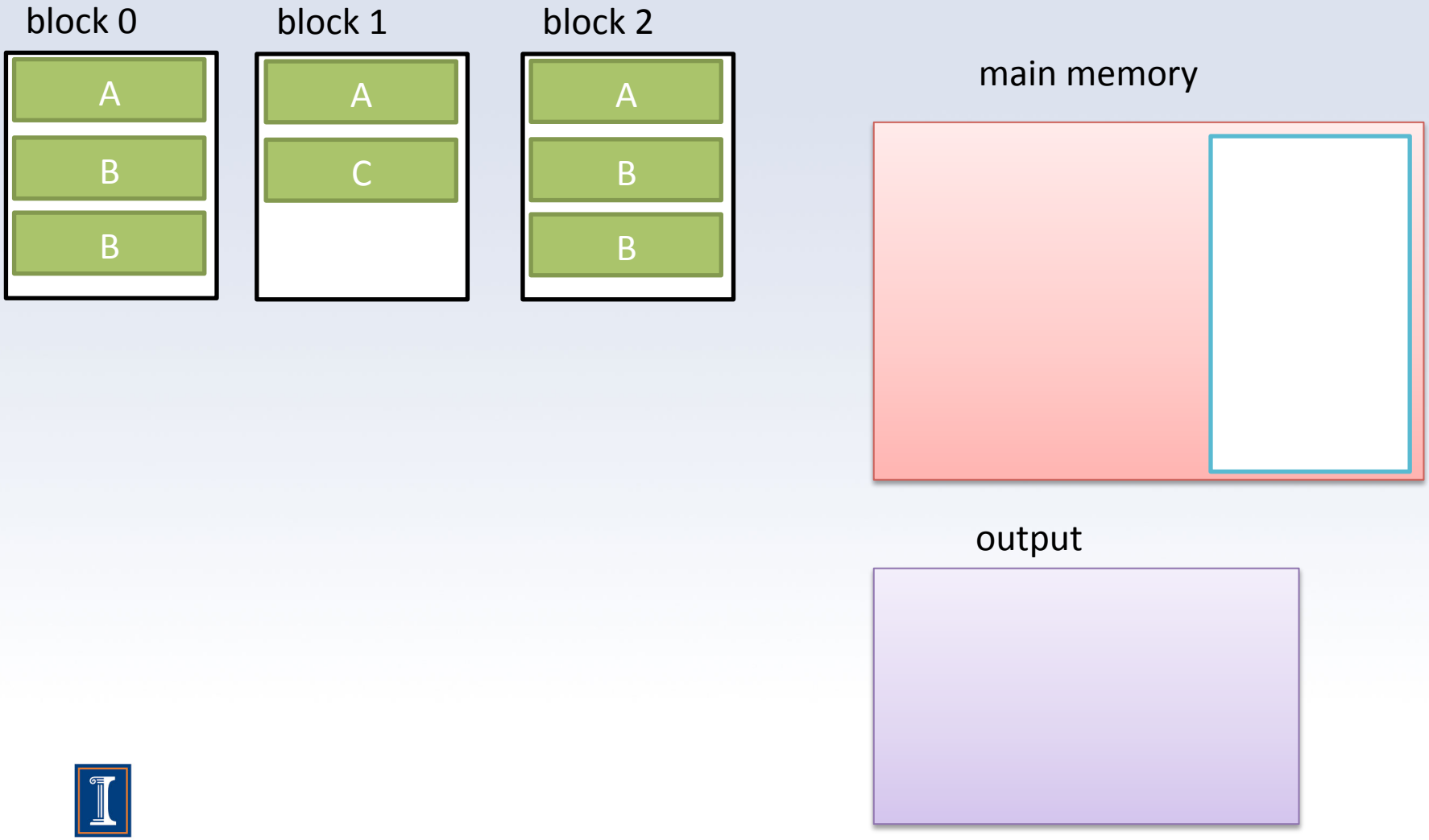
Example: <'G'


block 0
Q
X
F
block 1
A
R
block 2
T
V
L
block 3
O
Q
main memory
O
Q
output
F
A

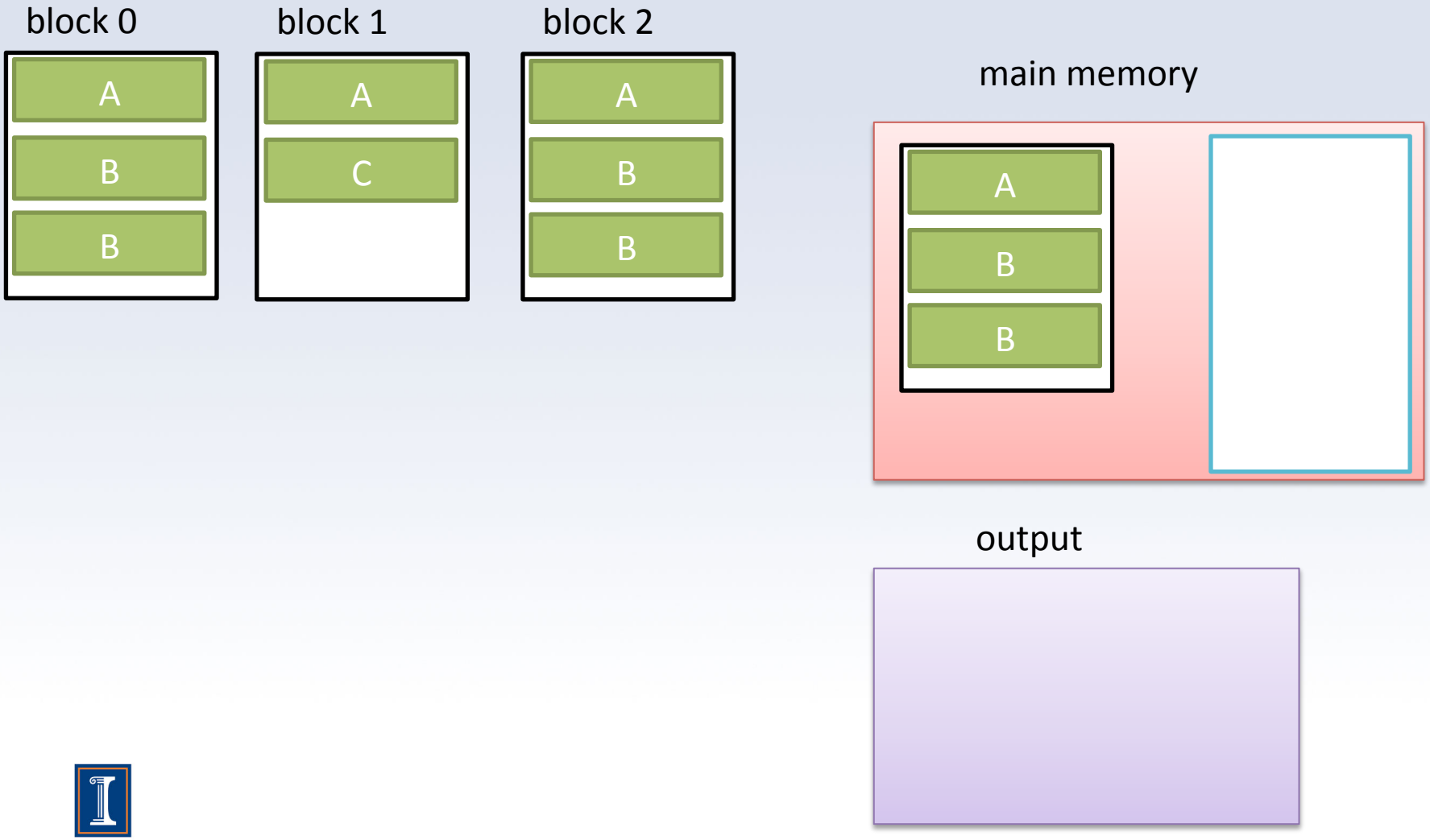
Unary Full-Relational

- For δ and γ, need to keep a *dictionary* in memory
  - Track what values we've seen so far
  - Also track aggregation results for γ
- Cost: B(R)
  - Assumes:
    - $B(\delta(R)) \leq M$
    - $B(\gamma(R)) \leq M$

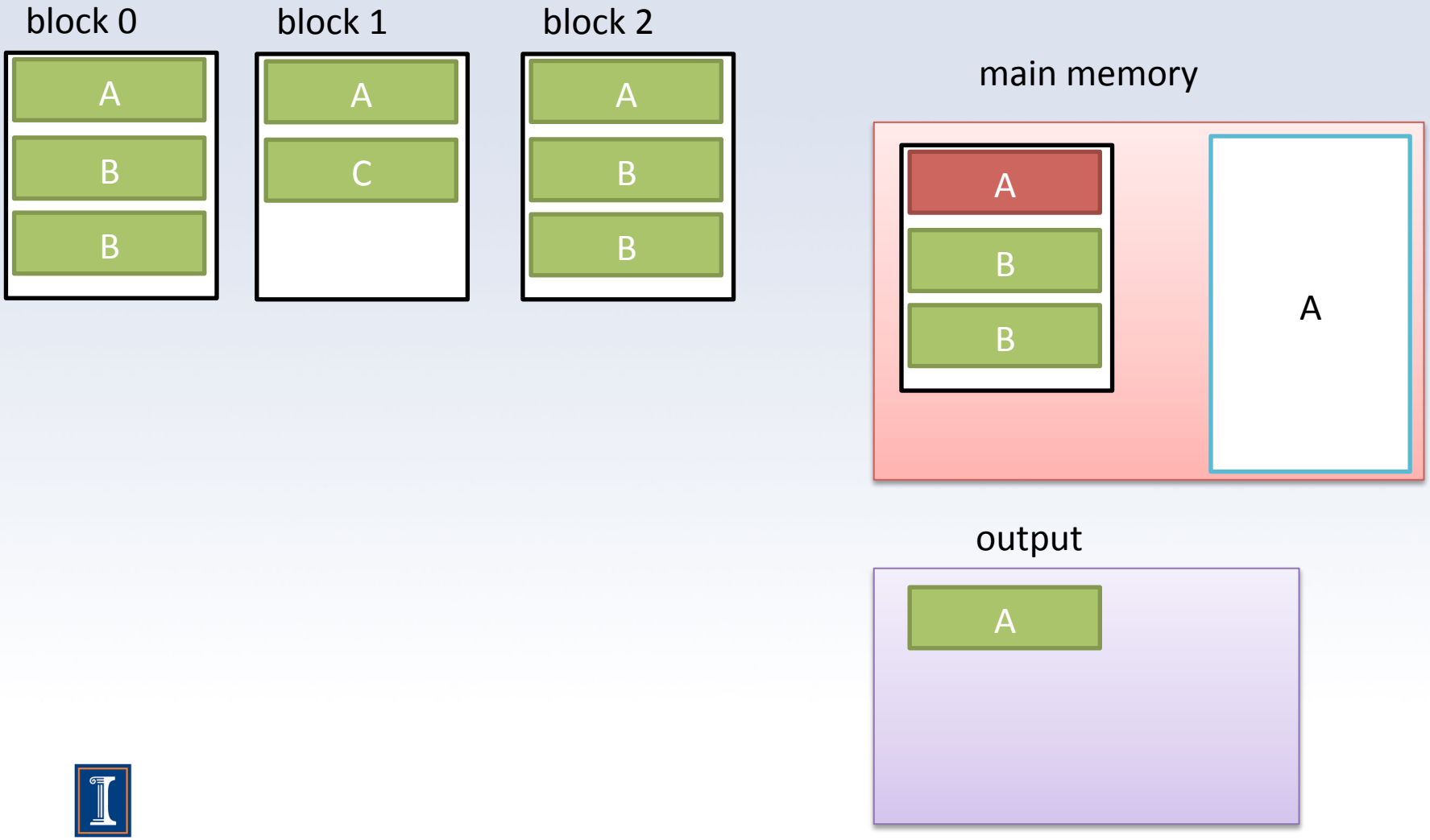
Example: δ



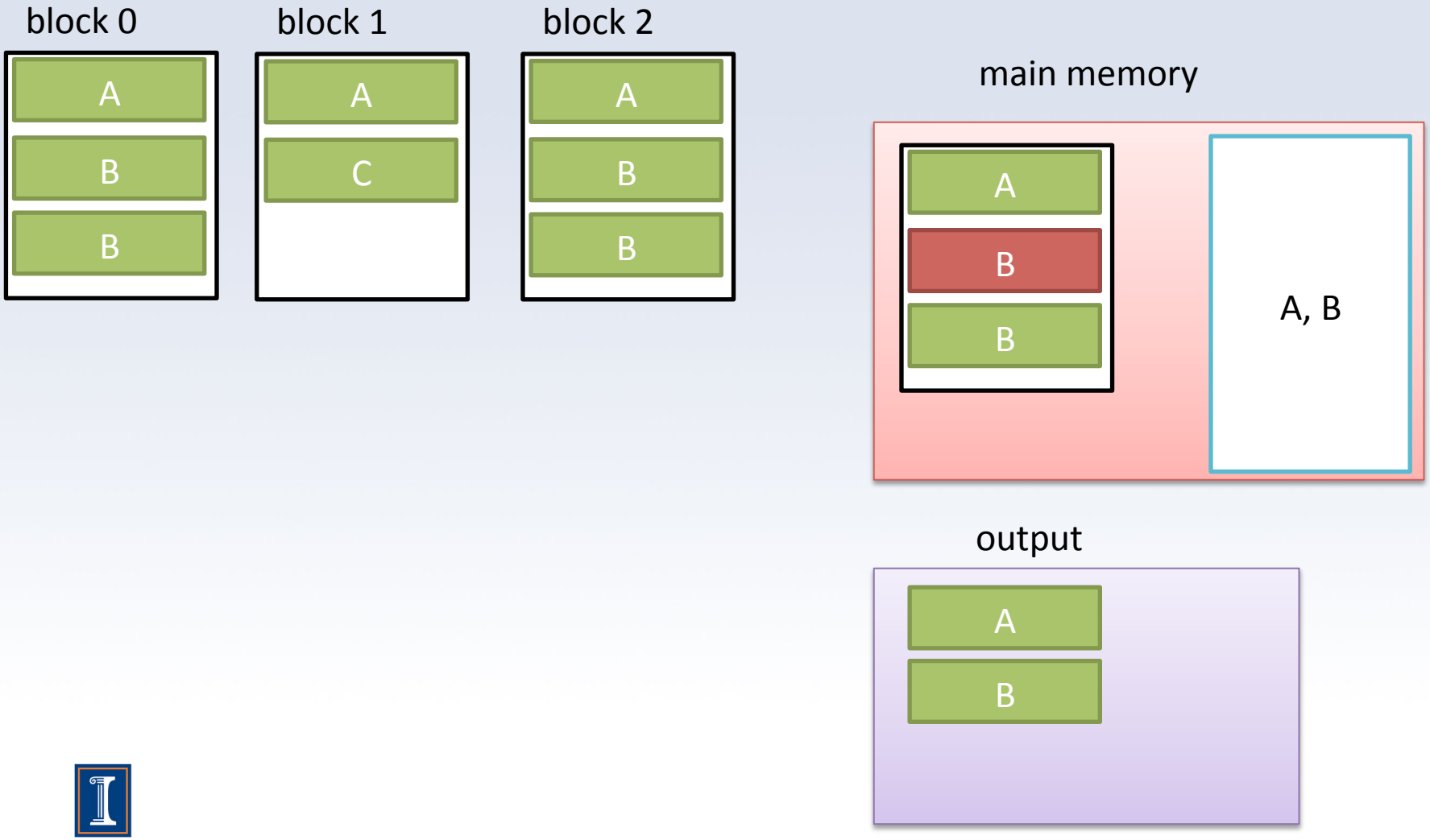
Example: δ



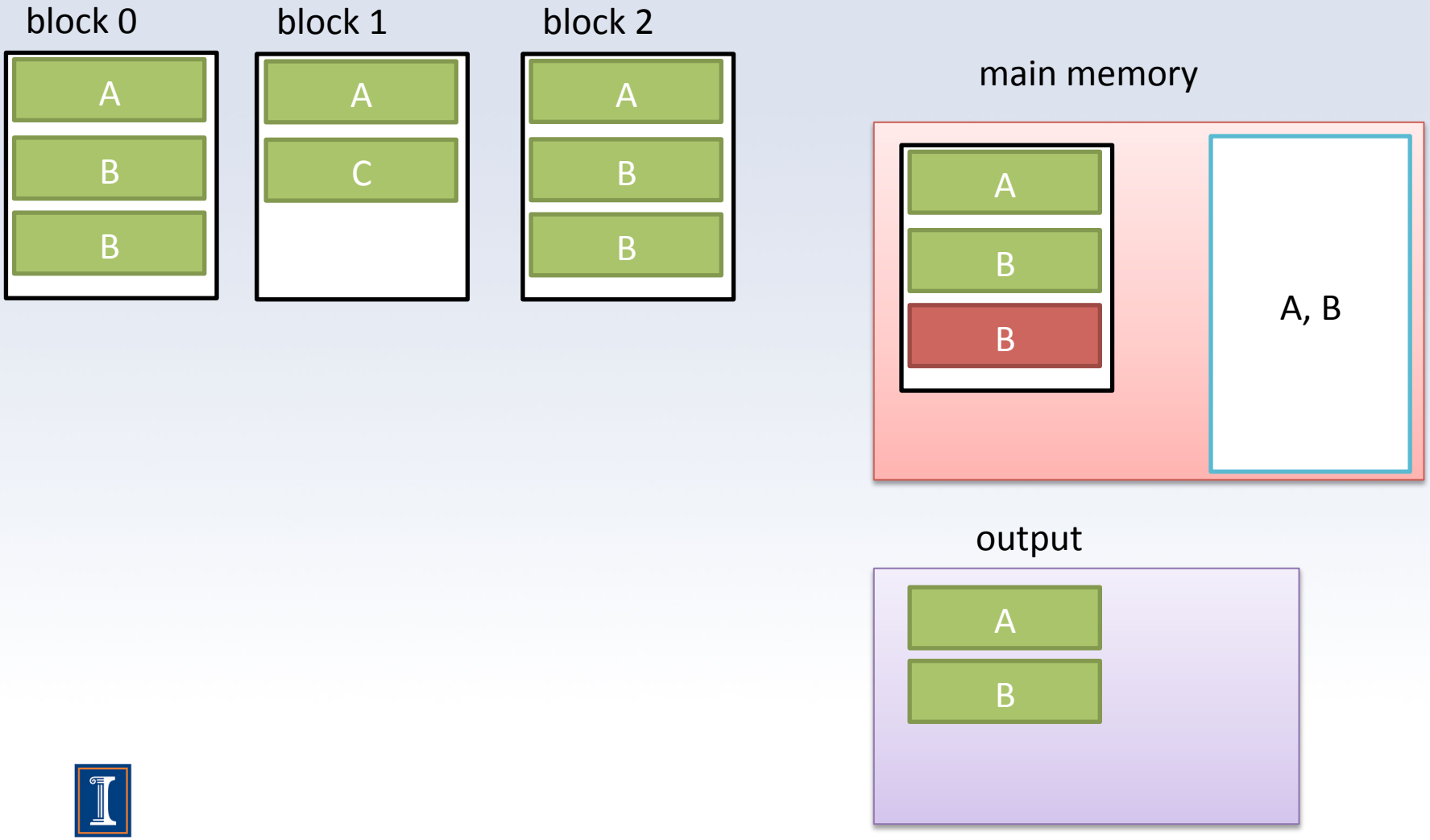
Example: δ



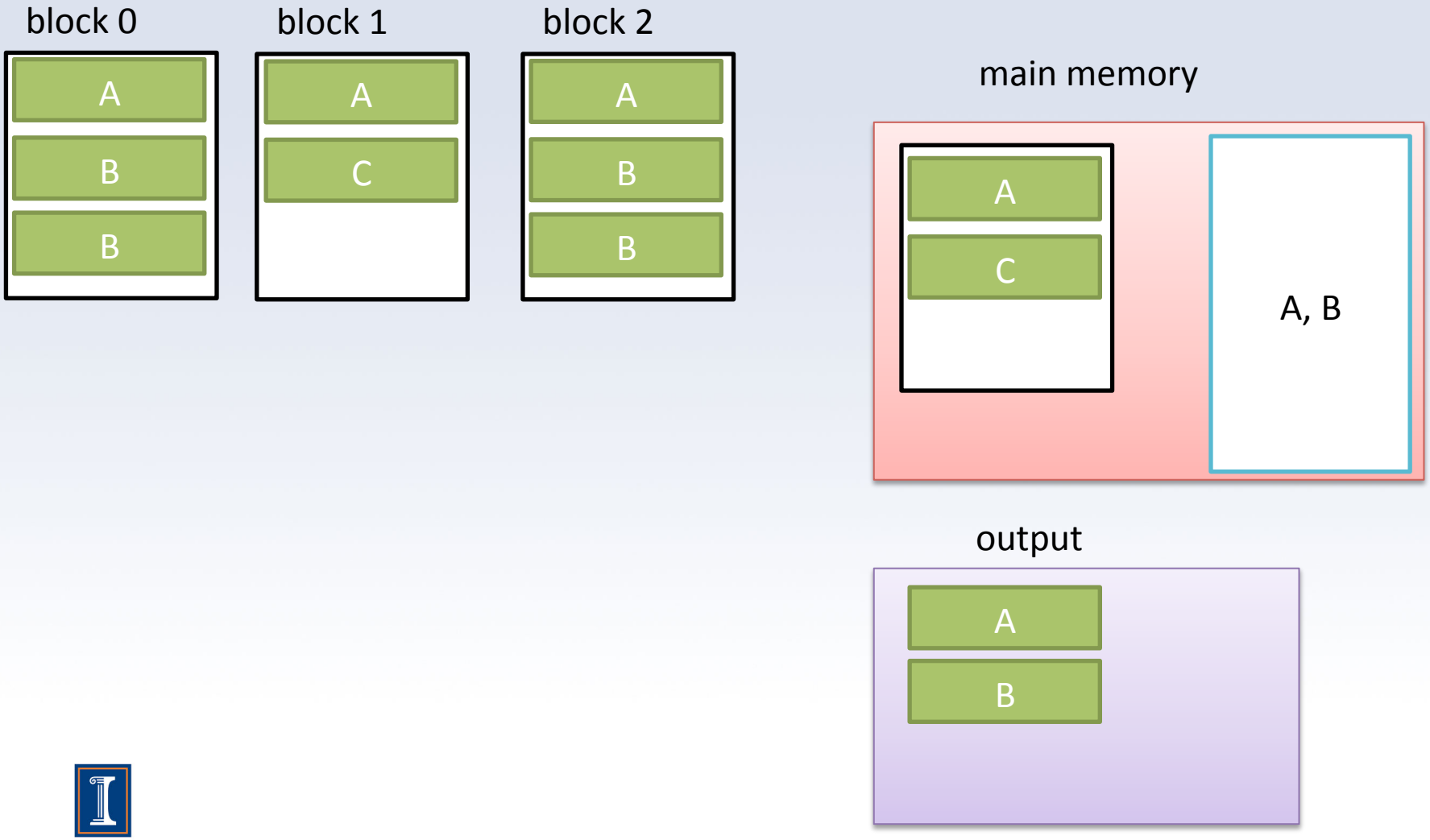
Example: δ



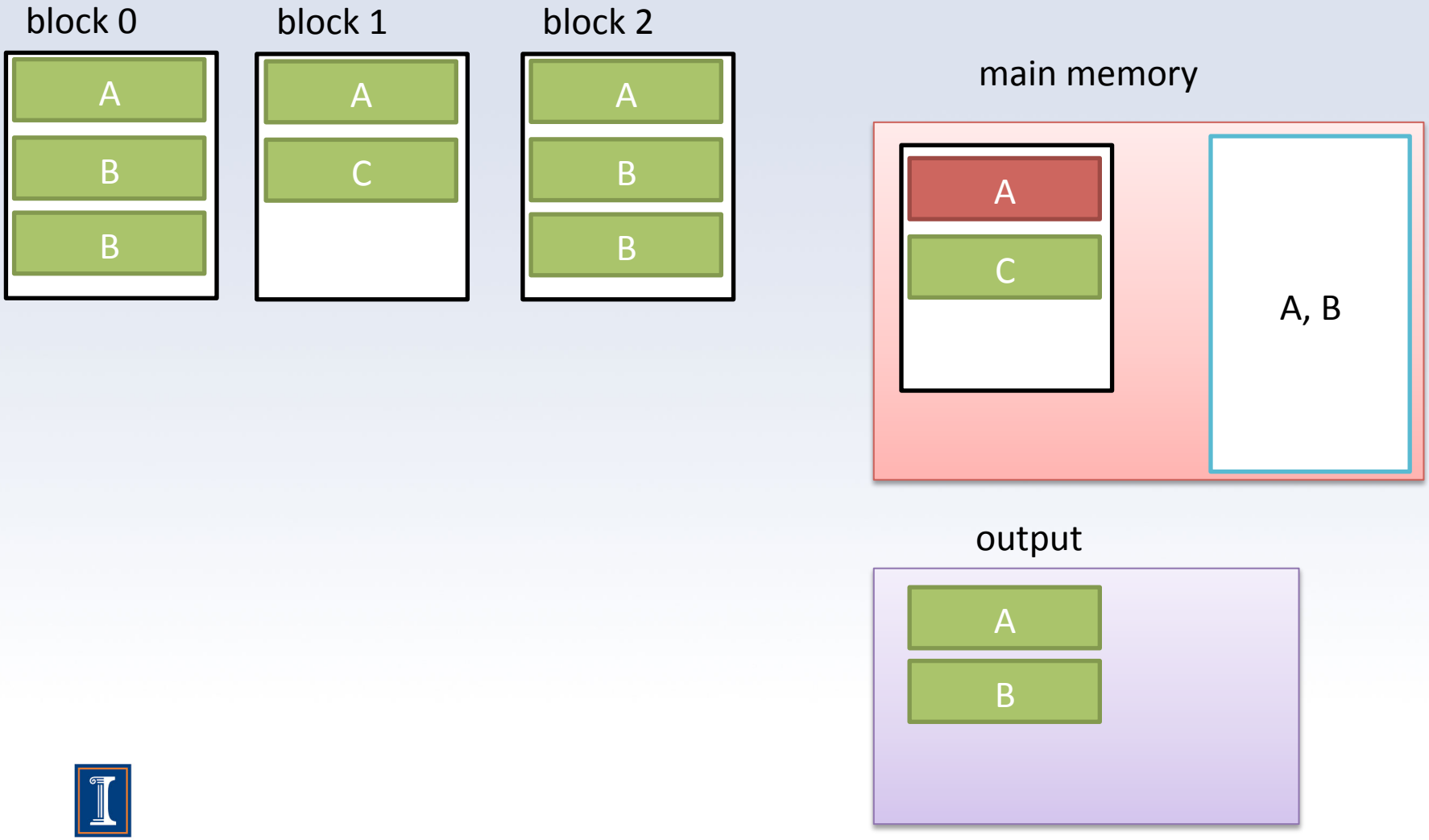
Example: δ



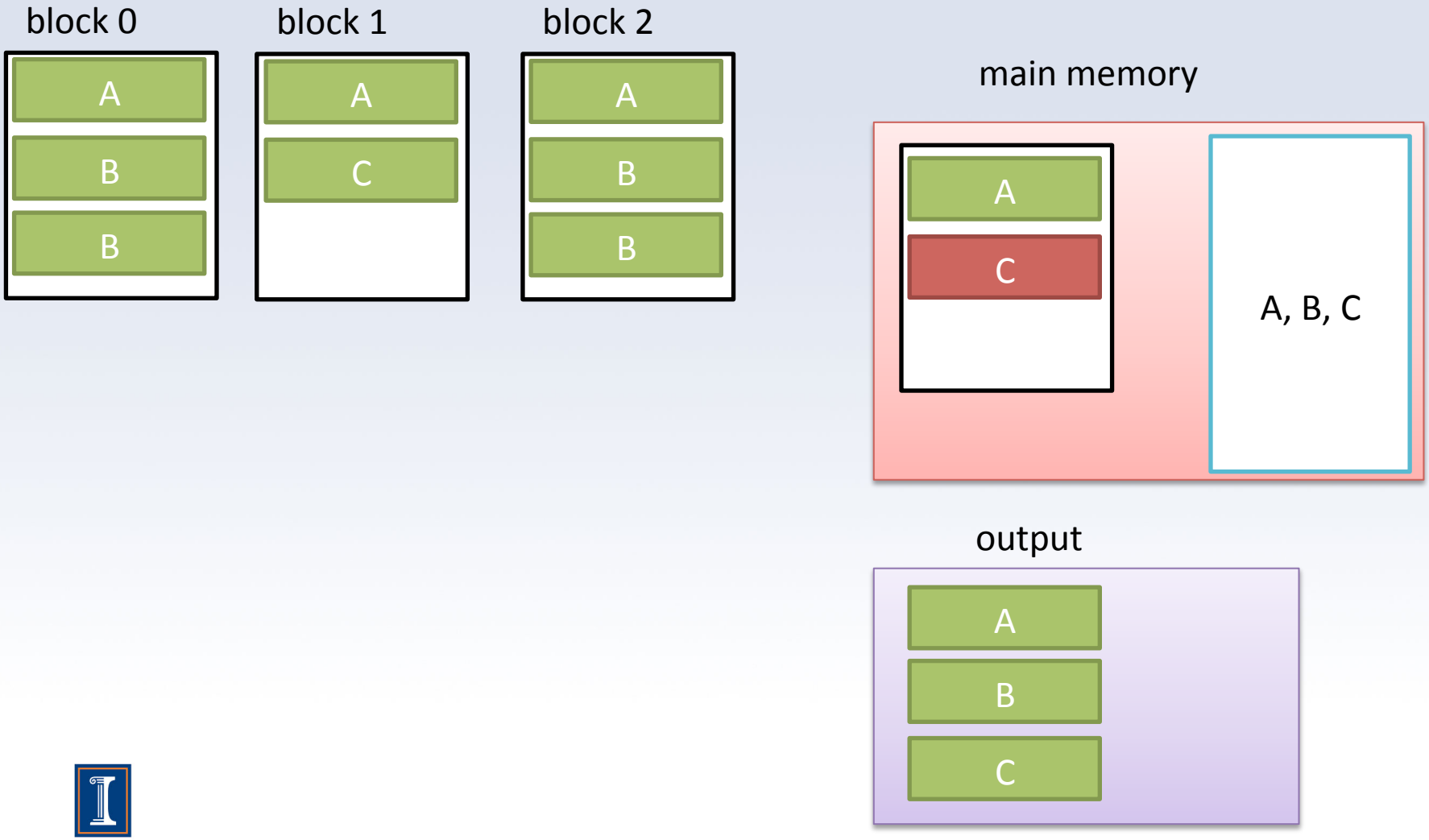
Example: δ



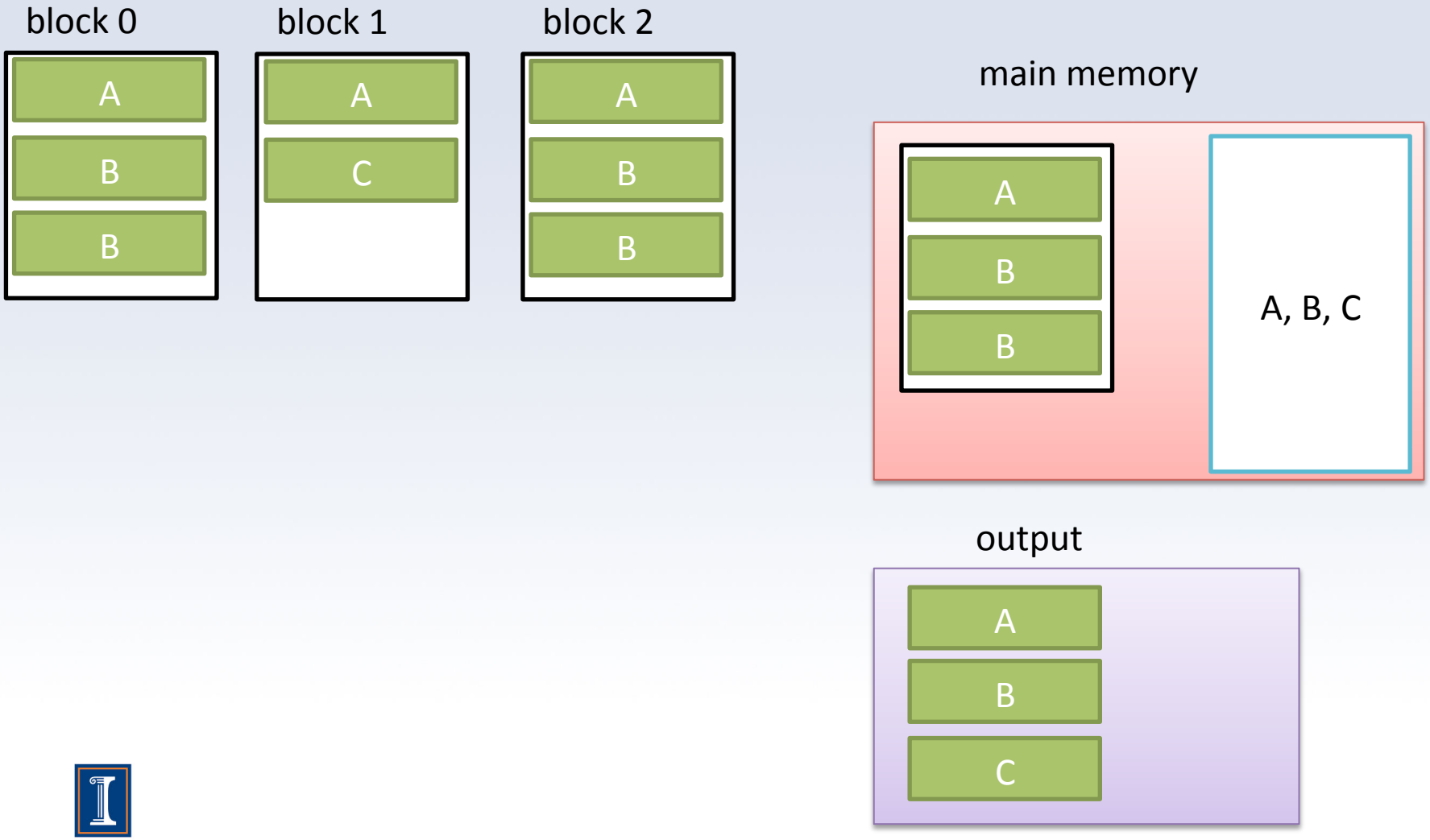
Example: δ



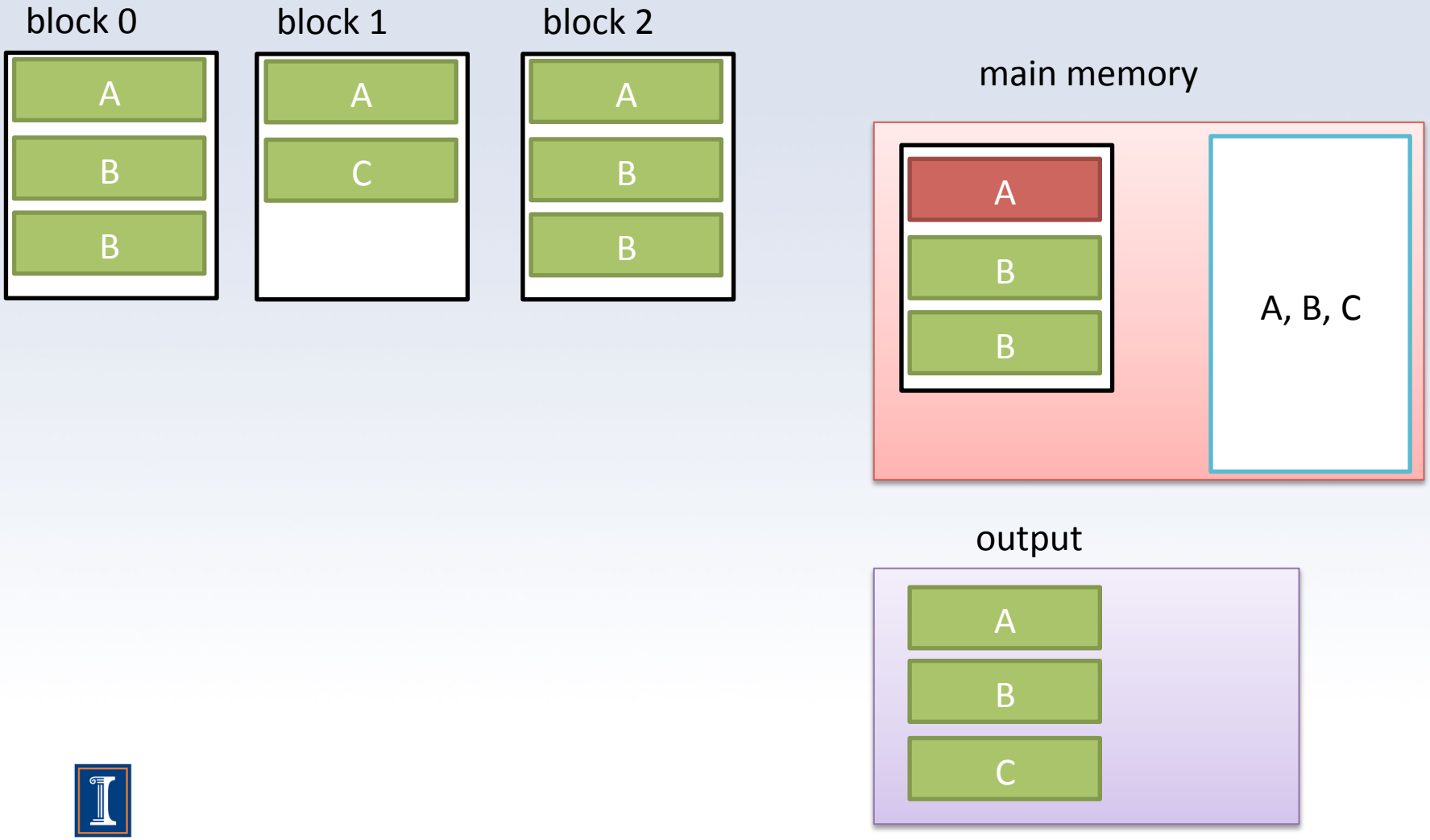
Example: δ


block 0
A
B
B
block 1
A
C
block 2
A
B
B
main memory
A
C
A, B, C
output
A
B
C

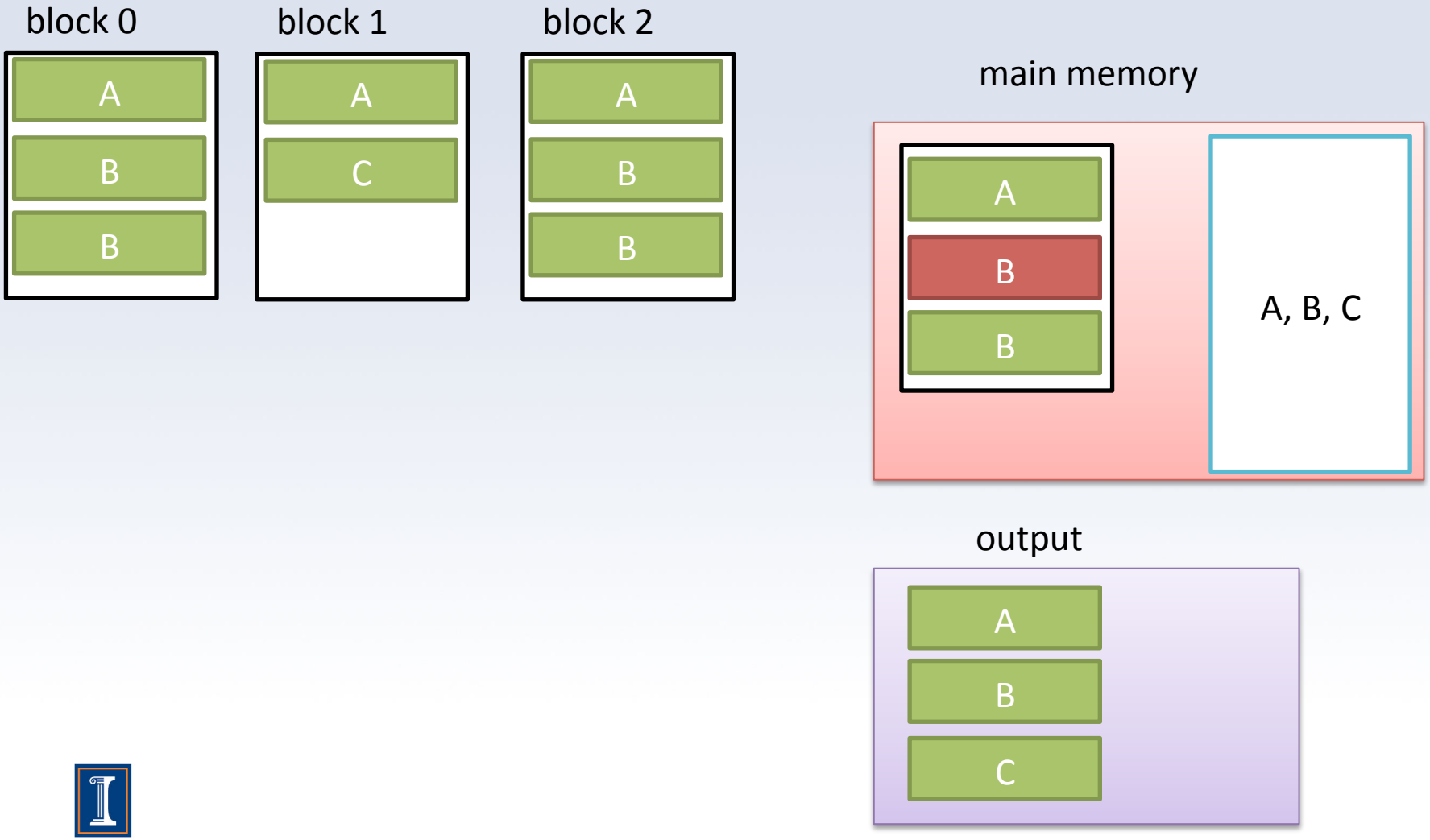
Example: δ


block 0
A
B
B
block 1
A
C
block 2
A
B
B
main memory
A
B
B
A, B, C
output
A
B
C

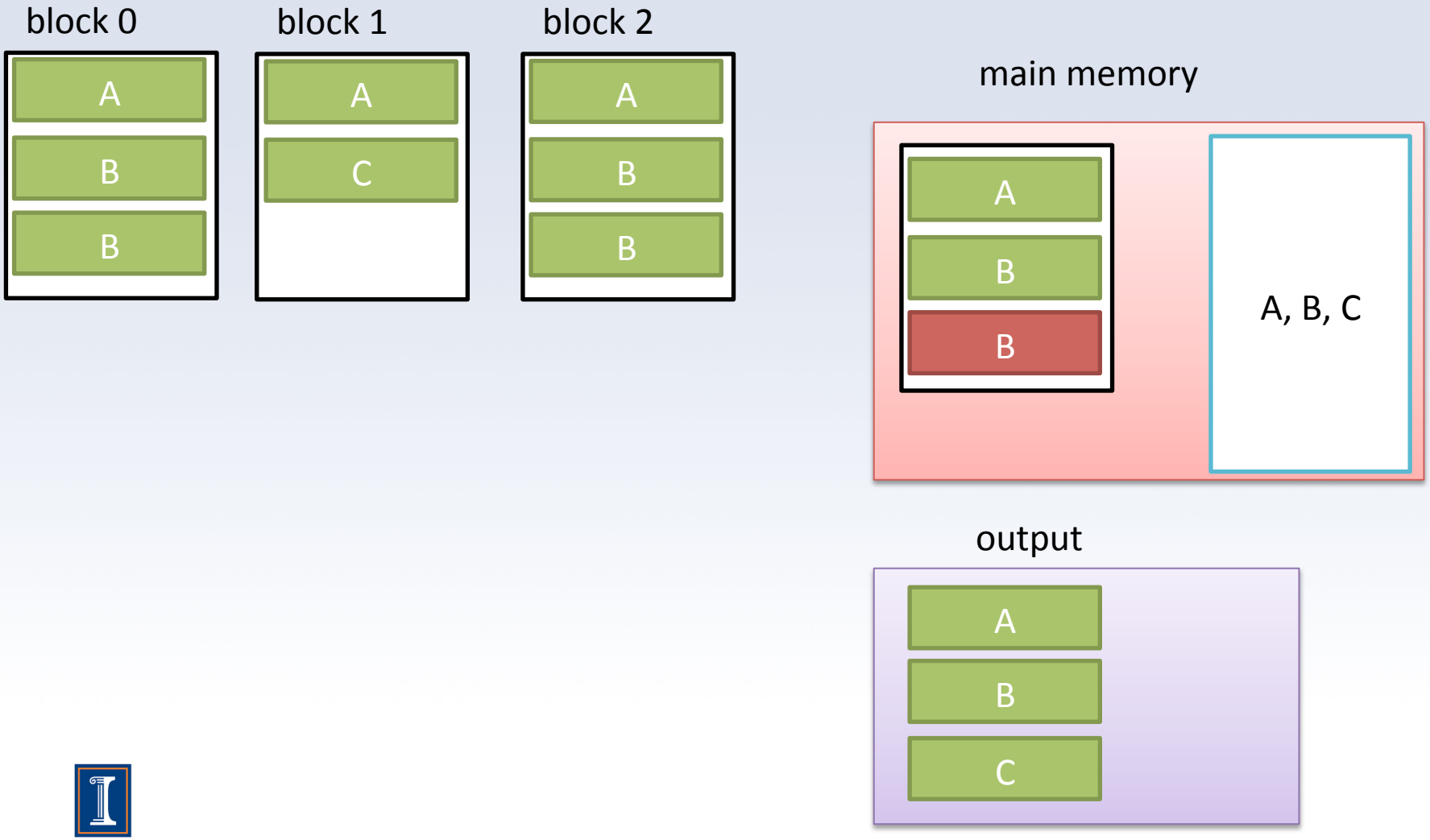
Example: δ


block 0
A
B
B
block 1
A
C
block 2
A
B
B
main memory
A
B
B
A, B, C
output
A
B
C

Example: δ


block 0
A
B
B
block 1
A
C
block 2
A
B
B
main memory
A
B
B
A, B, C
output
A
B
C

Example: δ


block 0
A
B
B
block 1
A
C
block 2
A
B
B
main memory
A
B
B
A, B, C
output
A
B
C

Binary Operations

- To do $\cup$, $\cap$, -, $\bowtie$, $\times$ in one pass:
  - We need one of the relations to fit in memory
  - $\min(B(R),B(S))<M$
- Assume S is smaller
- For set operations, assume $\delta(R)$ and $\delta(S)$ are already computed

Binary Operations

- ∩
  - read δ(S) into memory (M-1 buffers)
  - build dictionary
  - table scan δ(R) (1 buffer)
    - looking up each record in dictionary
    - output if entry in dictionary

Binary Operations

- ∪
  - read δ(S) into memory (M-1 buffers)
  - build dictionary and output δ(S)
  - table scan δ(R) (1 buffer)
    - look up each record in dictionary
    - output if entry *not* in dictionary

Binary Operations

- R-S
  - read δ(S) into memory (M-1 buffers)
  - build dictionary
  - table scan δ(R) (1 buffer)
    - look up each record in dictionary
    - output if entry *not* in dictionary

Binary Operations

- S-R
  - read δ(S) into memory (M-1 buffers)
  - build dictionary
  - table scan δ(R) (1 buffer)
    - look up each record in dictionary
    - if found, delete entry from dictionary
  - output remaining entries

Binary Operations

- ×
  - read S into memory (M-1 buffers)
  - table scan R (1 buffer)
    - for each tuple in S
      - pair it with the current tuple in R
      - output the pairing

Binary Operations

- ⋈
  - read S into memory (M-1 buffers)
  - build dictionary for join attributes
  - table scan R (1 buffer)
    - look up join attributes in dictionary
    - if found, output combined result

Example: R⋈S

block 0

A,x

B,y

B,z

block 1

A,g

C,g

block 2

A,a

B,b

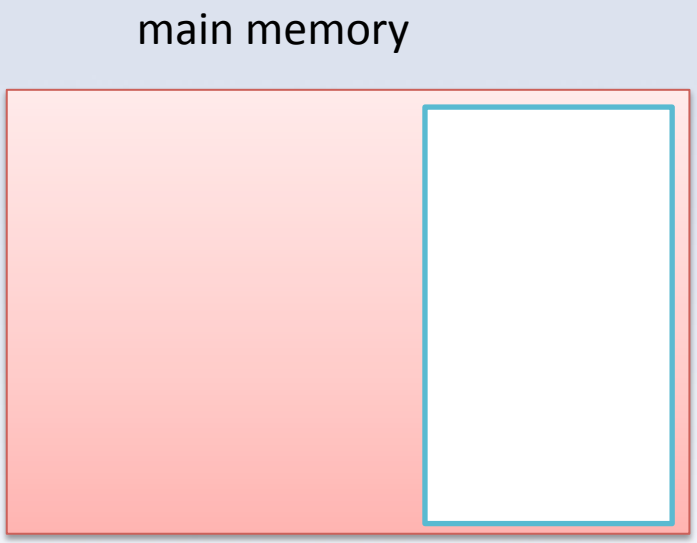
B,c

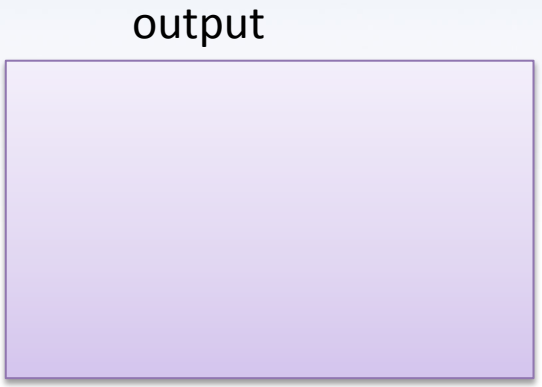
block 0

A,1

B,2

D,3





Example: R⋈S

block 0

A,x

B,y

B,z

block 1

A,g

C,g

block 2

A,a

B,b

B,c

main memory

A->1
B->2
D->3

block 0

A,1

B,2

D,3

output

Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

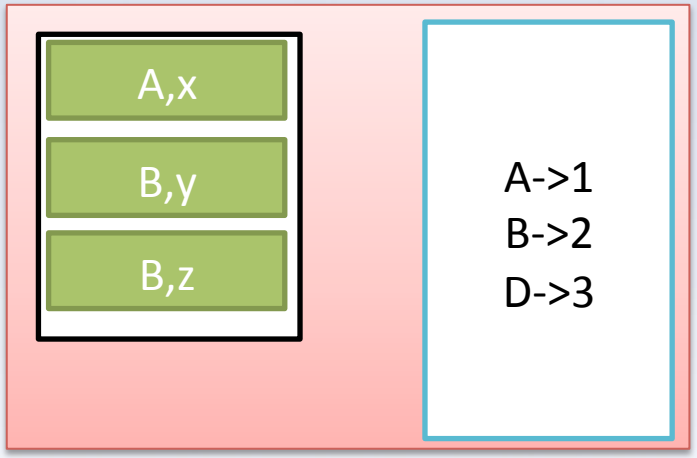
block 2

A,a
B,b
B,c

block 0

A,1
B,2
D,3

main memory



output

Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

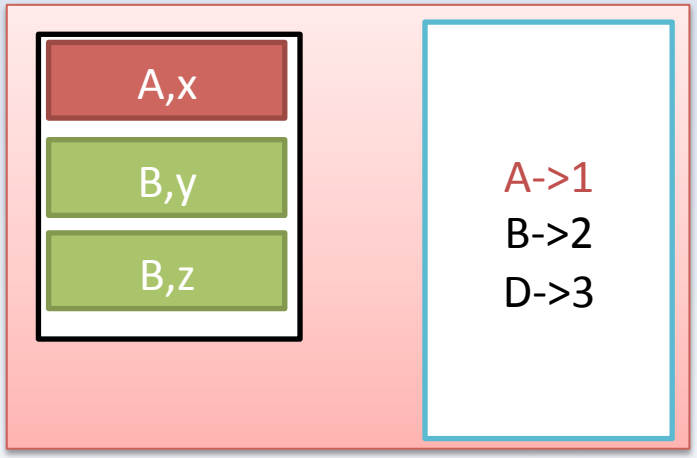
block 2

A,a
B,b
B,c

block 0

A,1
B,2
D,3

main memory

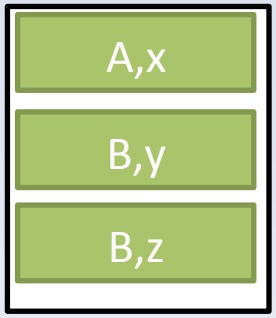


output

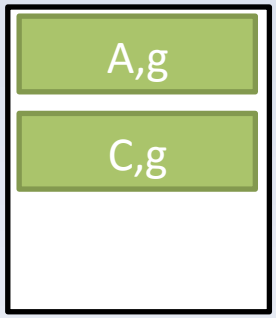
A,x,1

Example: R⋈S

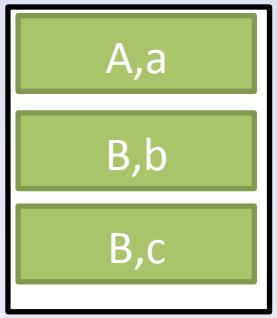
block 0



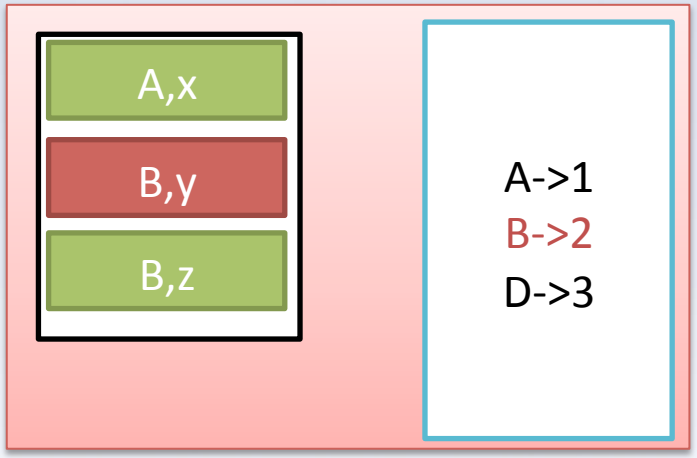
block 1



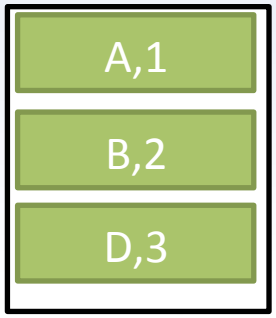
block 2



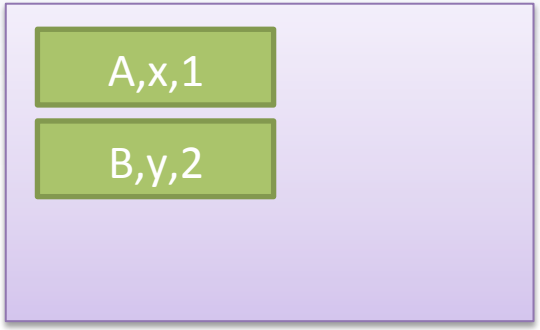
main memory



block 0



output



illinois.edu

Example: R⋈S

block 0

A,x

B,y

B,z

block 1

A,g

C,g

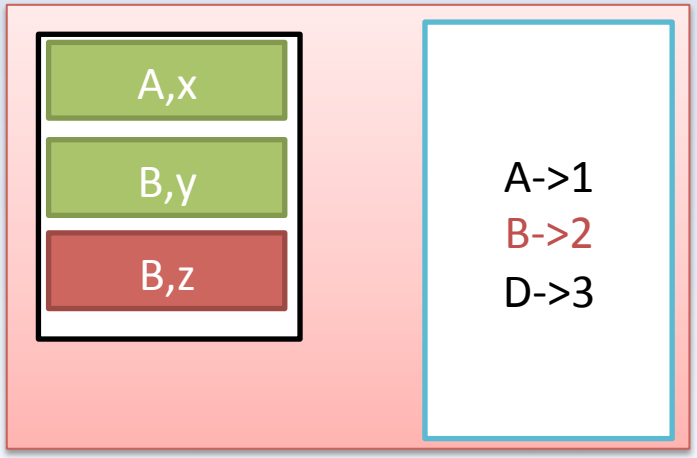
block 2

A,a

B,b

B,c

main memory



block 0

A,1

B,2

D,3

output

A,x,1

B,y,2

B,z,2

illinois.edu

Example: R⋈S

block 0

A,x

B,y

B,z

block 1

A,g

C,g

block 2

A,a

B,b

B,c

main memory

A,g

C,g

A->1
B->2
D->3

block 0

A,1

B,2

D,3

output

A,x,1

B,y,2

B,z,2

Example: R⋈S

block 0

A,x

B,y

B,z

block 1

A,g

C,g

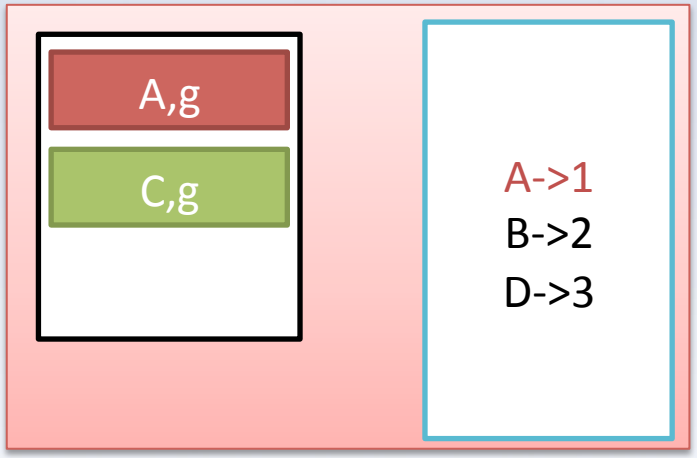
block 2

A,a

B,b

B,c

main memory



block 0

A,1

B,2

D,3

output

A,x,1

A,g,1

B,y,2

B,z,2

illinois.edu

Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

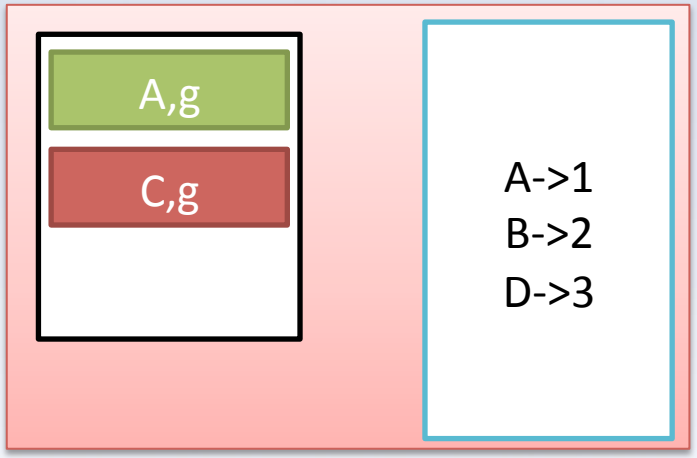
block 2

A,a
B,b
B,c

block 0

A,1
B,2
D,3

main memory



output

A,x,1	A,g,1
B,y,2	
B,z,2	

Example: R⋈S

block 0

A,x
B,y
B,z

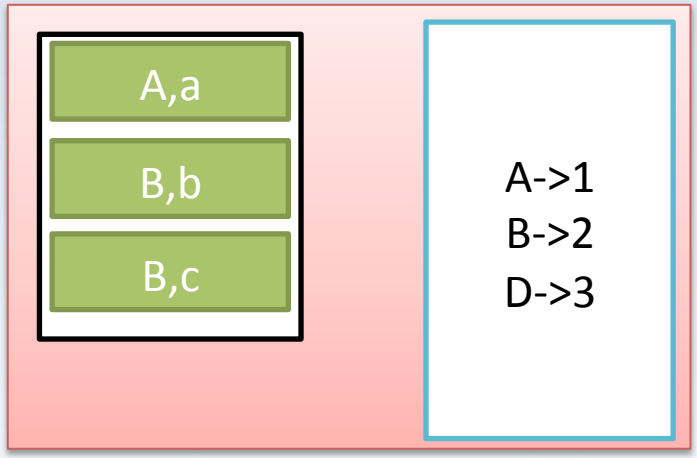
block 1

A,g
C,g

block 2

A,a
B,b
B,c

main memory



block 0

A,1
B,2
D,3

output

A,x,1	A,g,1
B,y,2	
B,z,2	

Example: R⋈S

block 0

A,x
B,y
B,z

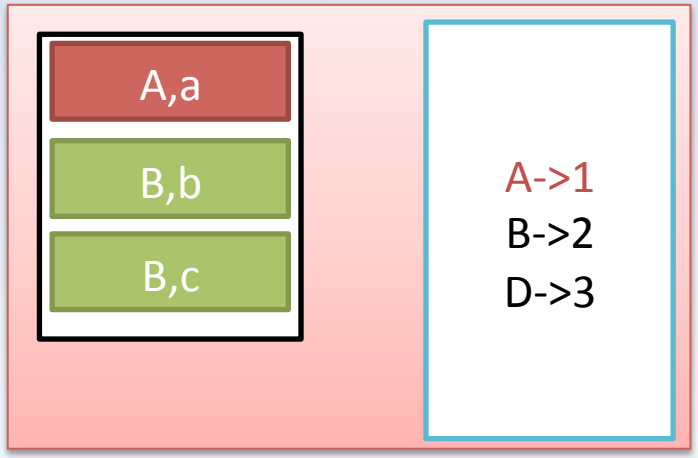
block 1

A,g
C,g

block 2

A,a
B,b
B,c

main memory



block 0

A,1
B,2
D,3

output

A,x,1	A,g,1
B,y,2	A,a,1
B,z,2	

Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

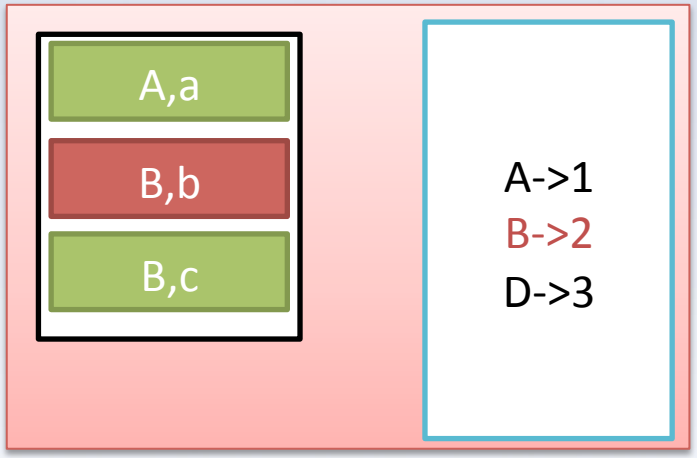
block 2

A,a
B,b
B,c

block 0

A,1
B,2
D,3

main memory



output

A,x,1	A,g,1
B,y,2	A,a,1
B,z,2	B,b,2

Example: R⋈S

block 0

A,x
B,y
B,z

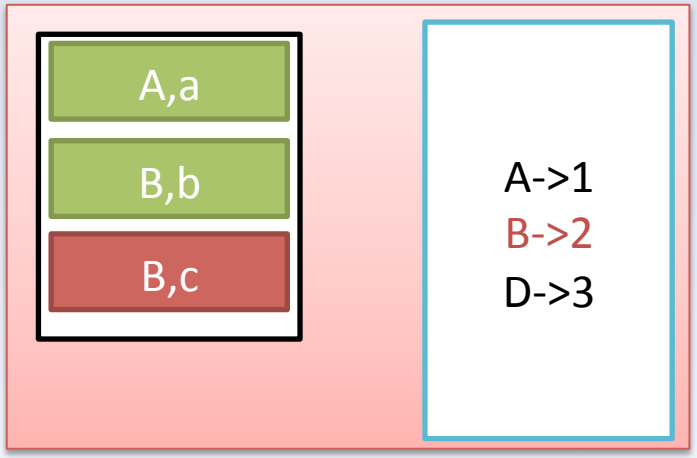
block 1

A,g
C,g

block 2

A,a
B,b
B,c

main memory



block 0

A,1
B,2
D,3

output

A,x,1	A,g,1
B,y,2	A,a,1
B,z,2	B,b,2
	B,c,2

illinois.edu

Binary Single Pass Cost

- Required memory:
  - Need all blocks of R or S to fit into our buffer pool
  - min(B(R),B(S))
- I/O operations
  - Read smaller relation into memory takes B(S)
  - Table scan larger relation requires B(R)
  - Overall: B(R)+B(S)

Nested Loop Joins

- Execute the join with two for loops
- One relation is only read once, other is read multiple times

Pseudocode: tuple based

```
for s in S:
  for r in R:
    t=join(r,s)
    if(t):
      output t
```

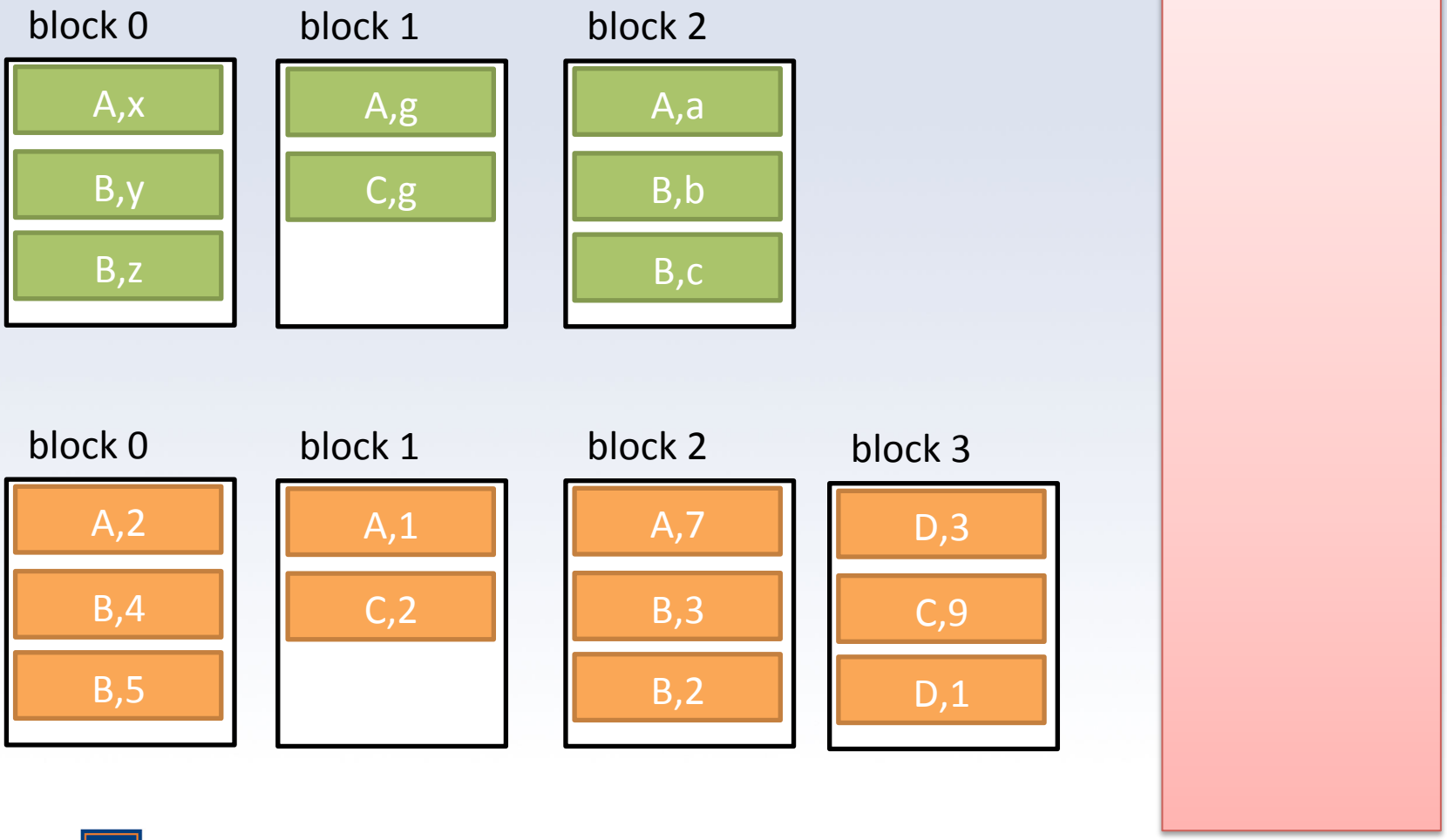
Nested Loop-Join

- That pseudocode ignores the buffer/block model of data
- Nested-loop join is really implemented:
  - Outer loop fills M-1 buffers with blocks from S
  - Inner loop fills last buffer with a block from R
- S is read once
- R is read B(S)/(M-1) times

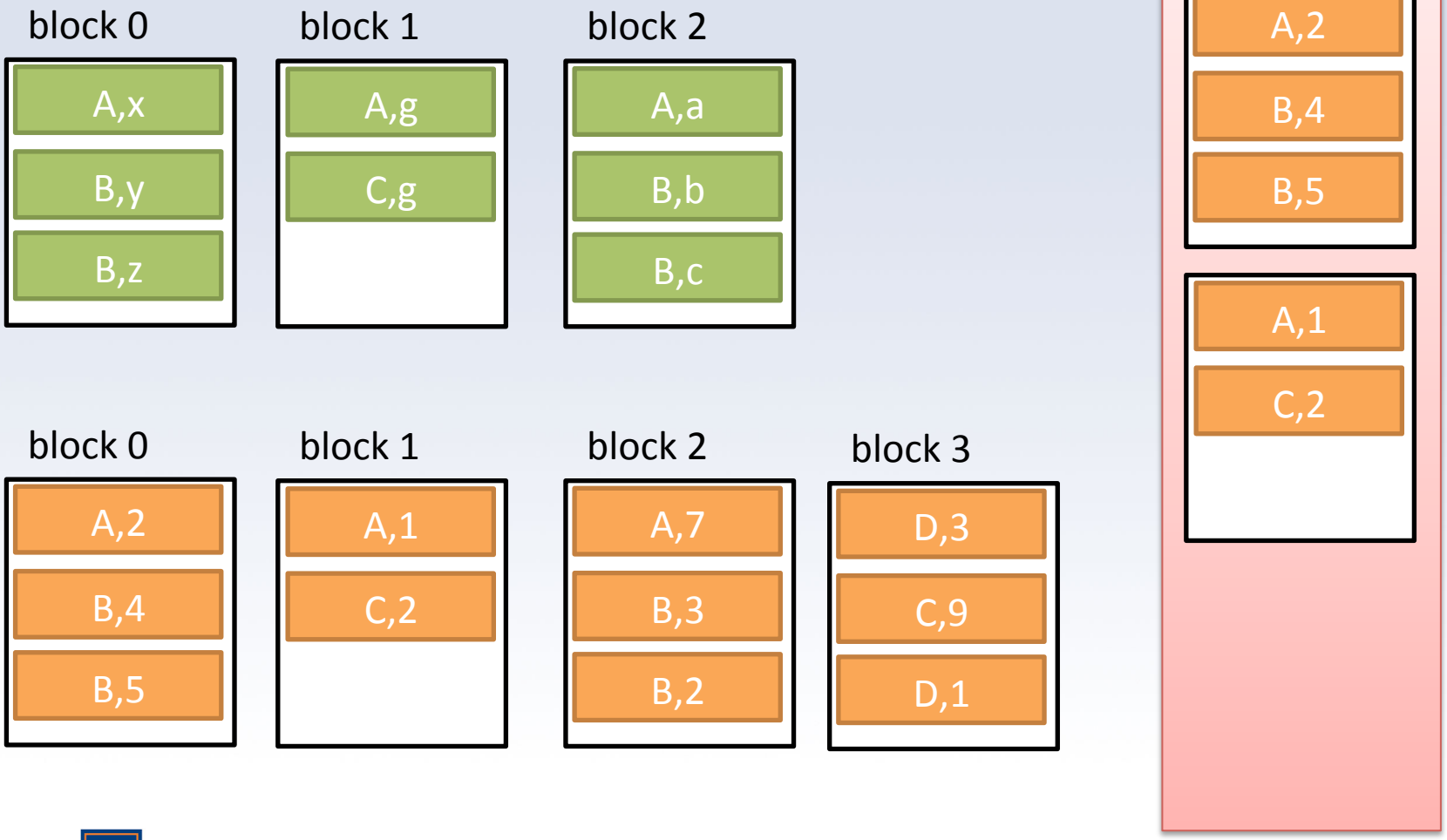
Pseudocode: Block-based

```
for (M-1) blocks bs in S:
  for block br in R:
    for s in bs:
      for r in br:
        t=join(r,s)
        if(t):
          output t
```

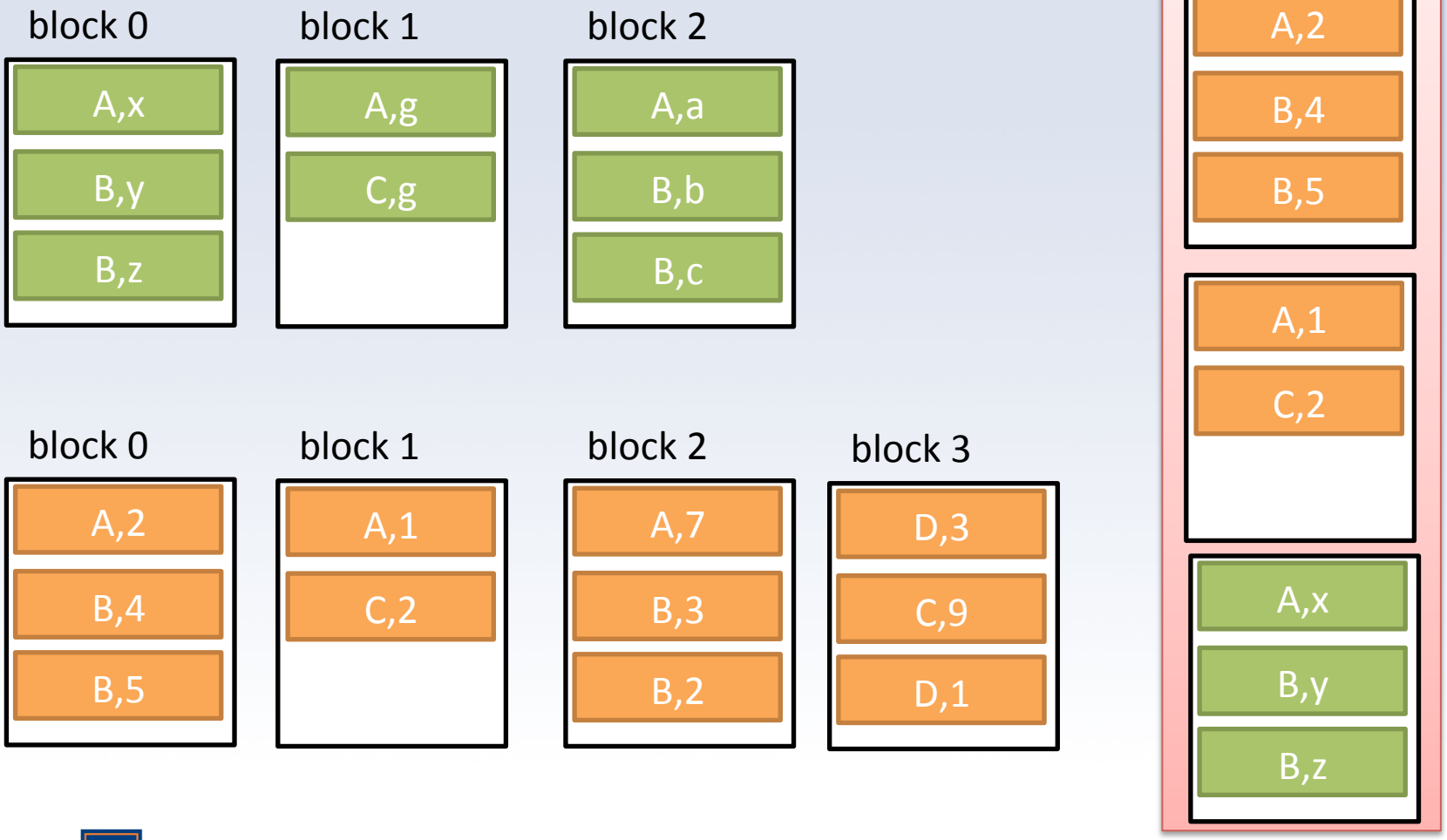
Example: R⋈S


block 0
A,x
B,y
B,z
block 1
A,g
C,g
block 2
A,a
B,b
B,c
block 0
A,2
B,4
B,5
block 1
A,1
C,2
block 2
A,7
B,3
B,2
block 3
D,3
C,9
D,1

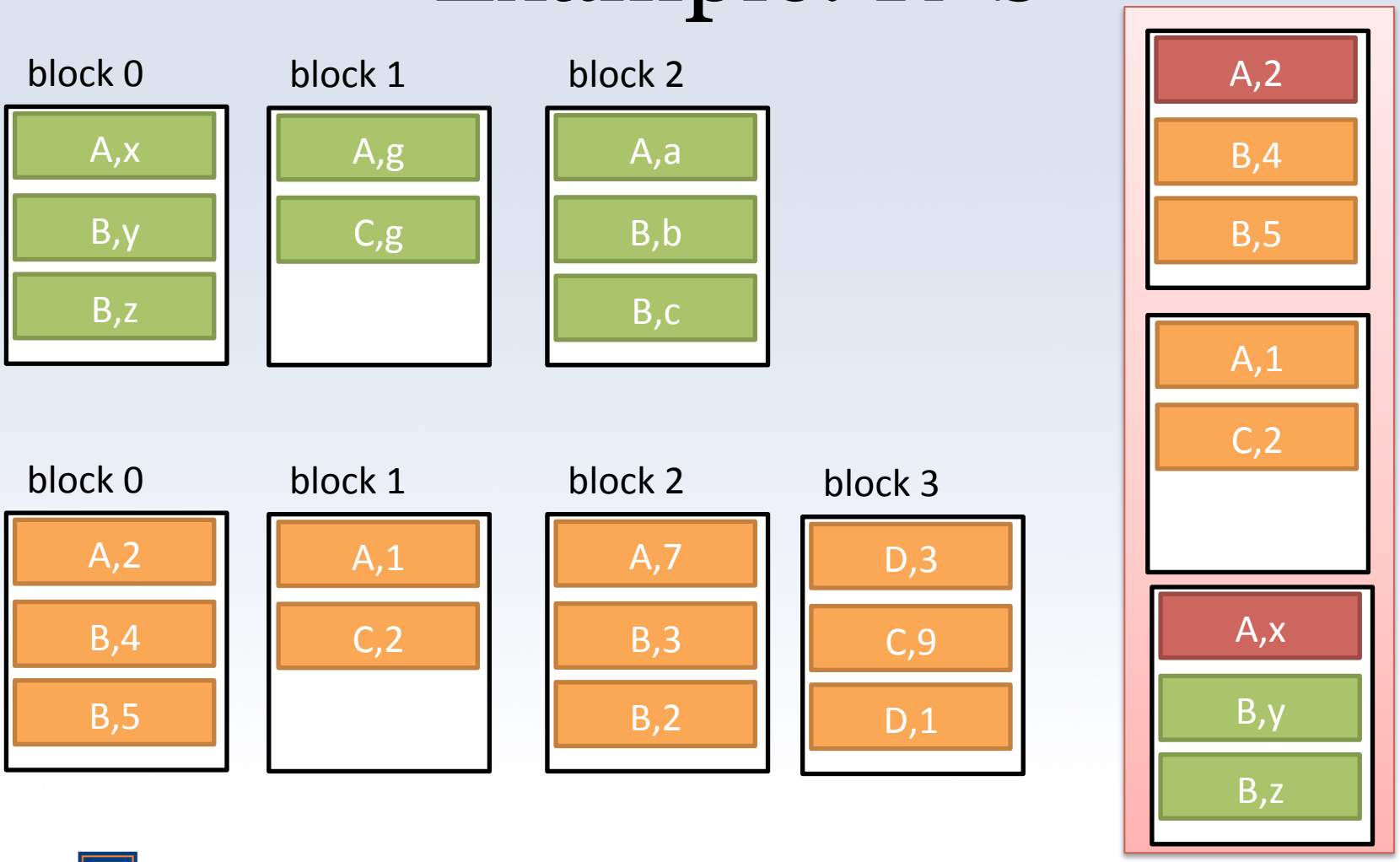
Example: R⋈S


block 0
A,x
B,y
B,z
block 1
A,g
C,g
block 2
A,a
B,b
B,c
block 0
A,2
B,4
B,5
block 1
A,1
C,2
block 2
A,7
B,3
B,2
block 3
D,3
C,9
D,1
A,2
B,4
B,5
A,1
C,2

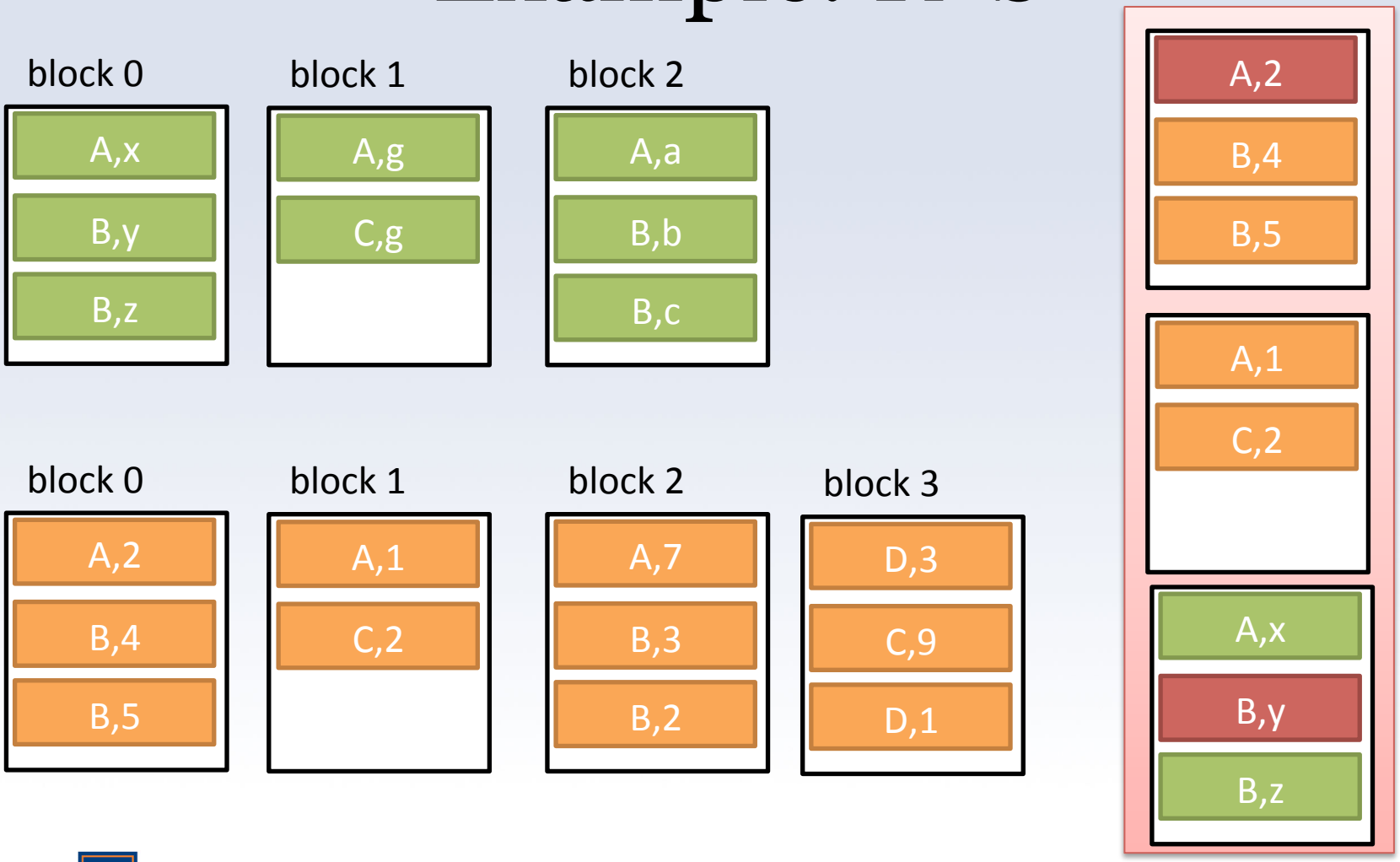
Example: R⋈S


block 0
A,x
B,y
B,z
block 1
A,g
C,g
block 2
A,a
B,b
B,c
block 0
A,2
B,4
B,5
block 1
A,1
C,2
block 2
A,7
B,3
B,2
block 3
D,3
C,9
D,1
A,2
B,4
B,5
A,1
C,2
A,x
B,y
B,z

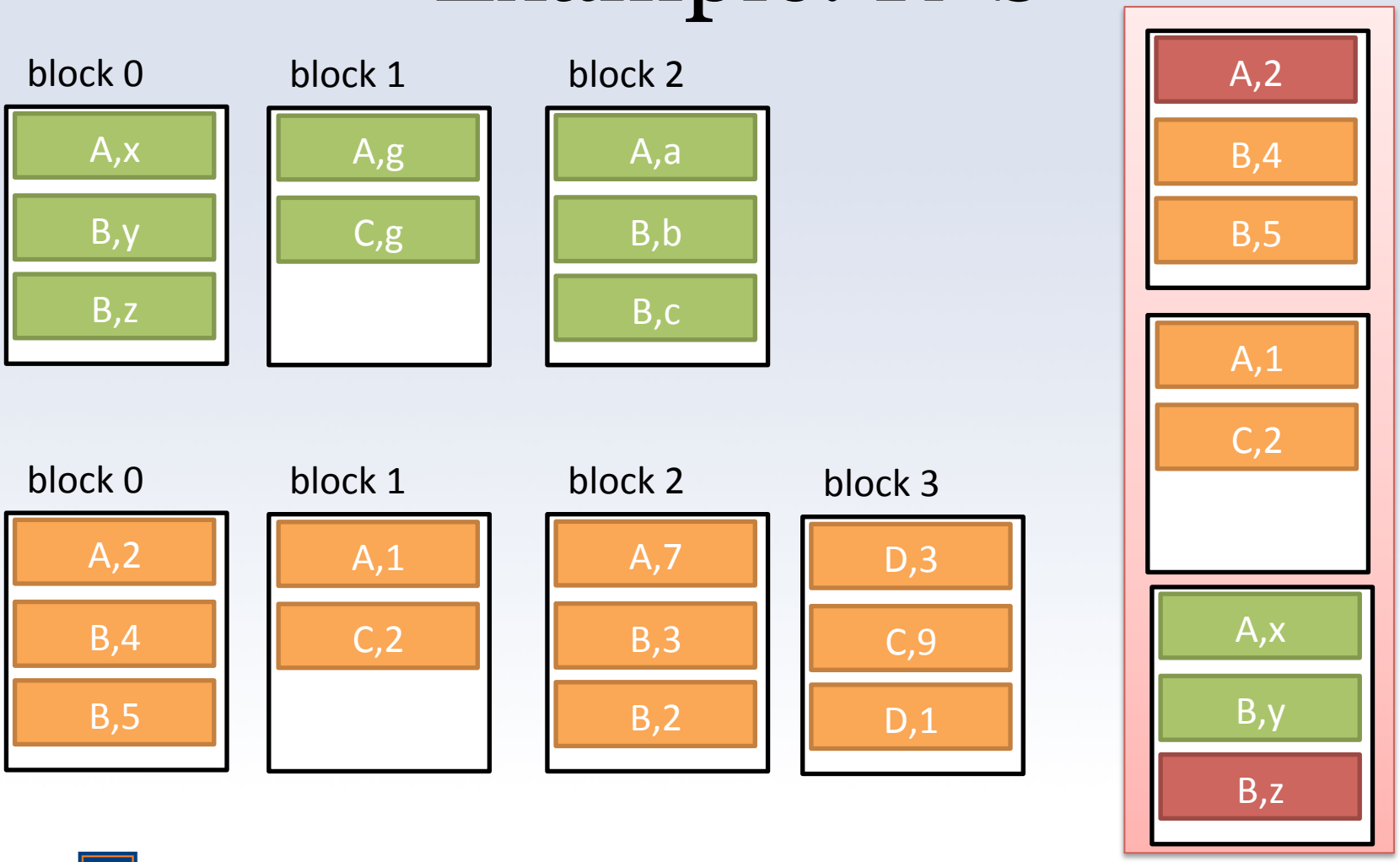
Example: R⋈S


block 0
A,x
B,y
B,z
block 1
A,g
C,g
block 2
A,a
B,b
B,c
block 0
A,2
B,4
B,5
block 1
A,1
C,2
block 2
A,7
B,3
B,2
block 3
D,3
C,9
D,1
A,2
B,4
B,5
A,1
C,2
A,x
B,y
B,z

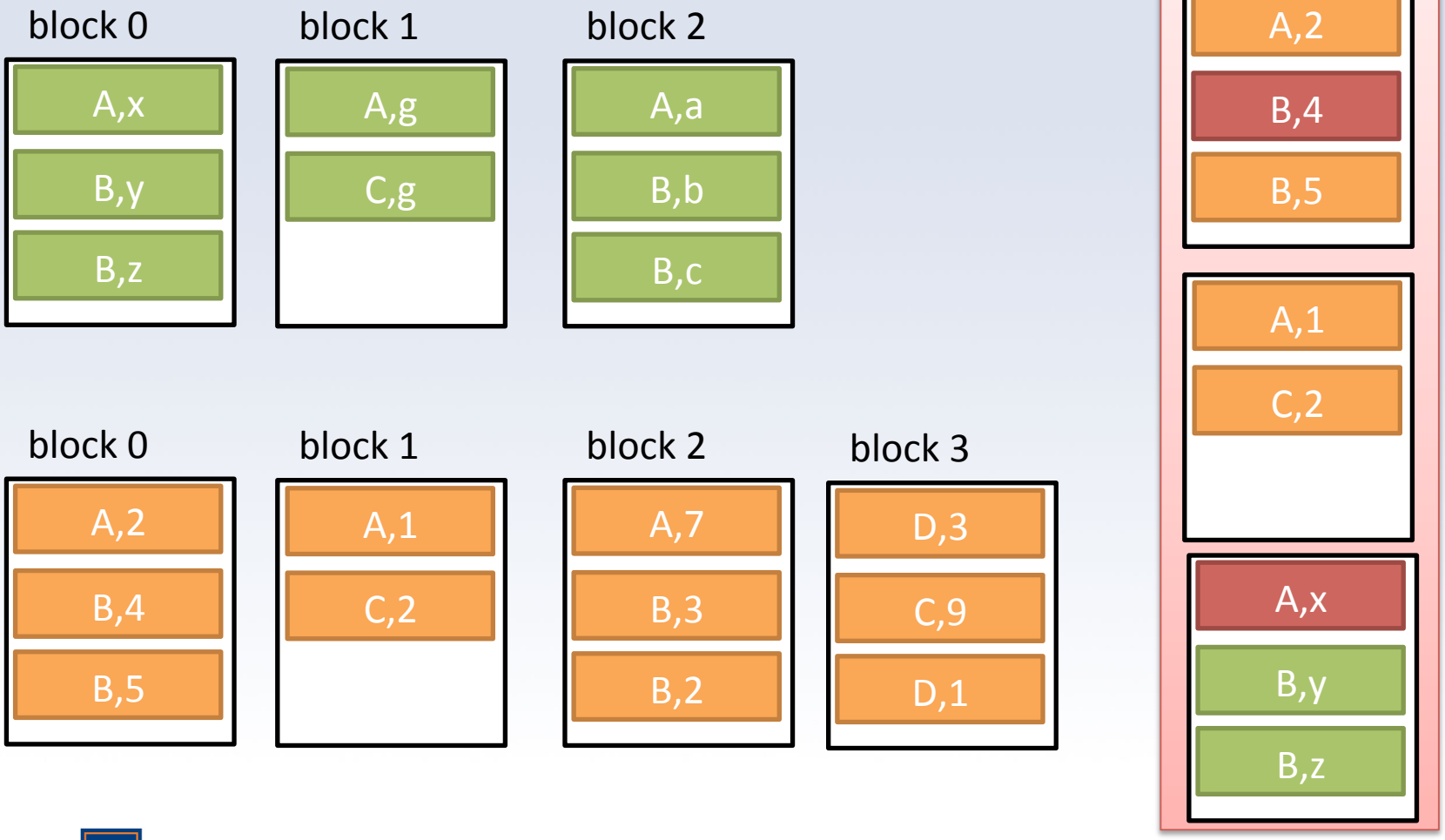
Example: R⋈S


block 0
A,x
B,y
B,z
block 1
A,g
C,g
block 2
A,a
B,b
B,c
block 0
A,2
B,4
B,5
block 1
A,1
C,2
block 2
A,7
B,3
B,2
block 3
D,3
C,9
D,1
A,2
B,4
B,5
A,1
C,2
A,x
B,y
B,z

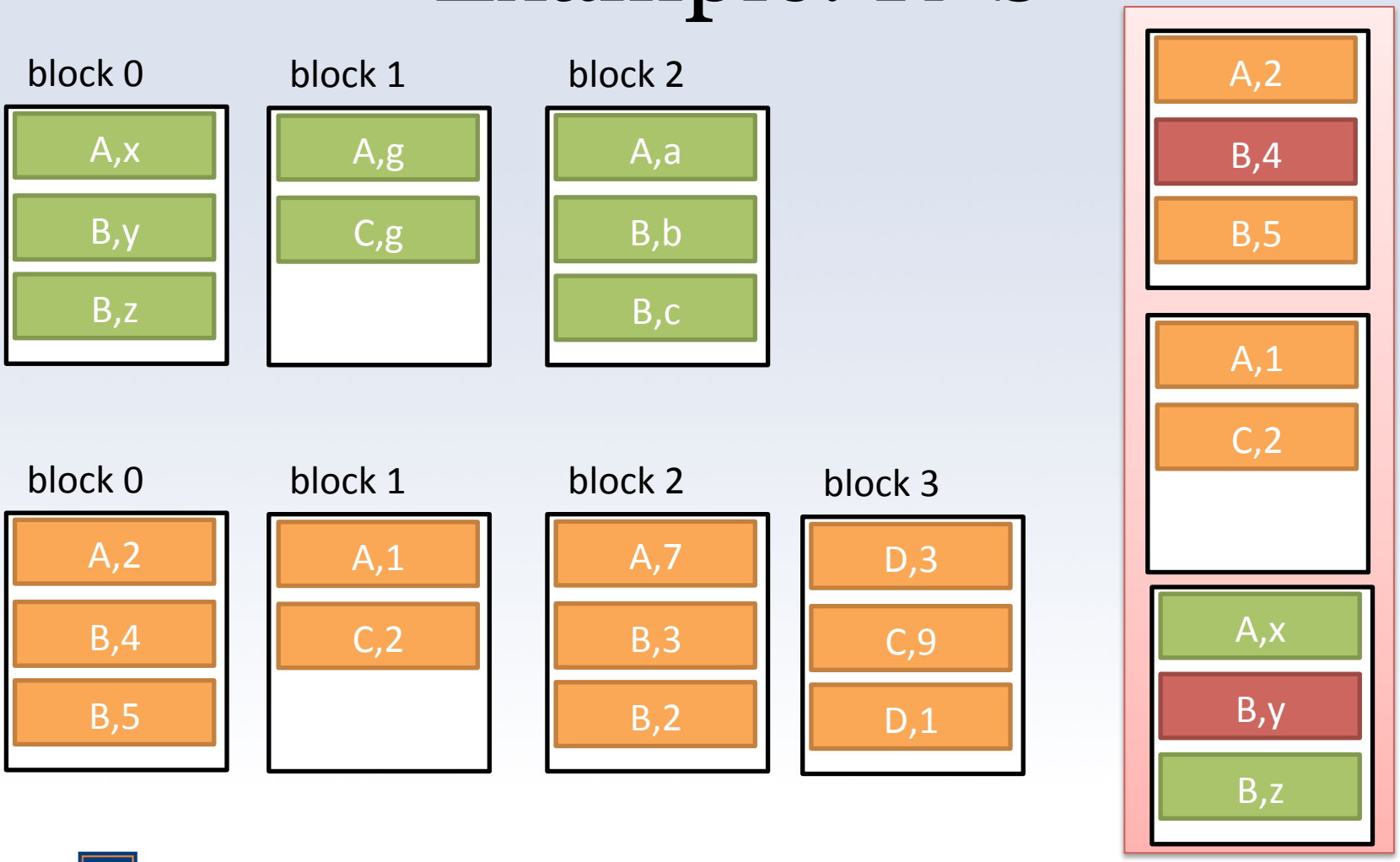
Example: R⋈S


block 0
A,x
B,y
B,z
block 1
A,g
C,g
block 2
A,a
B,b
B,c
block 0
A,2
B,4
B,5
block 1
A,1
C,2
block 2
A,7
B,3
B,2
block 3
D,3
C,9
D,1
A,2
B,4
B,5
A,1
C,2
A,x
B,y
B,z

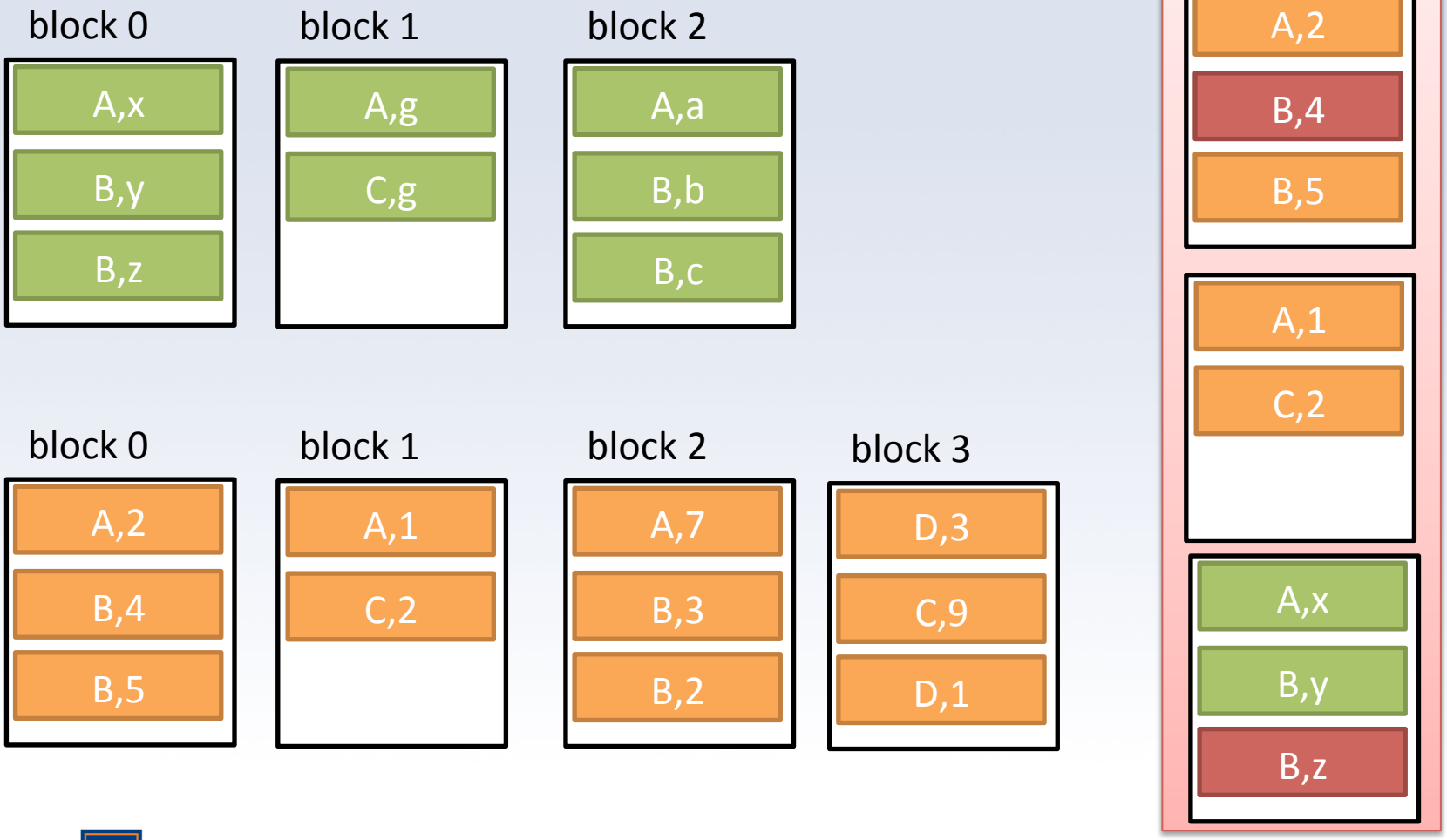
Example: R⋈S


block 0
A,x
B,y
B,z
block 1
A,g
C,g
block 2
A,a
B,b
B,c
block 0
A,2
B,4
B,5
block 1
A,1
C,2
block 2
A,7
B,3
B,2
block 3
D,3
C,9
D,1
A,2
B,4
B,5
A,1
C,2
A,x
B,y
B,z

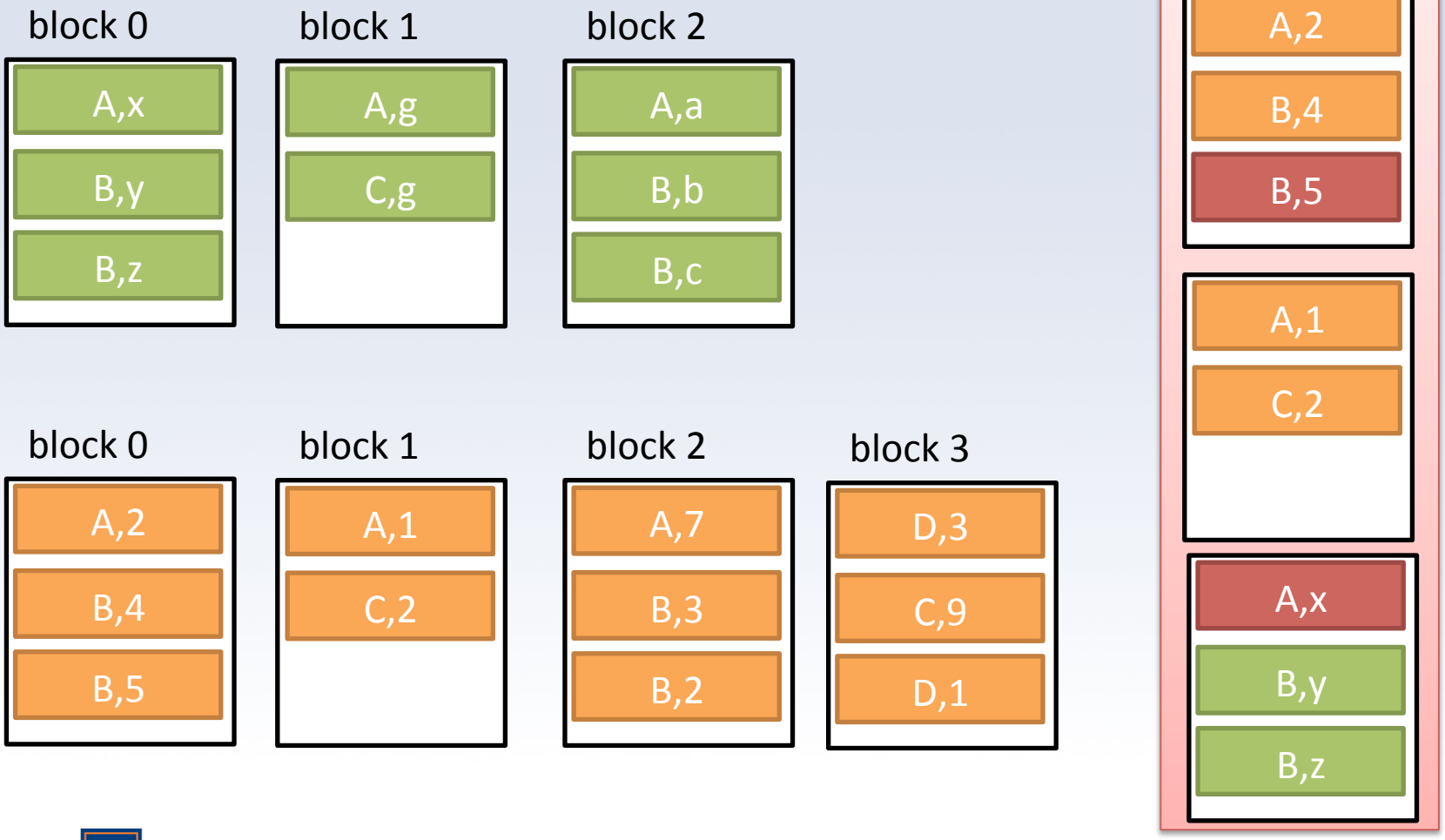
Example: R⋈S


block 0
A,x
B,y
B,z
block 1
A,g
C,g
block 2
A,a
B,b
B,c
block 0
A,2
B,4
B,5
block 1
A,1
C,2
block 2
A,7
B,3
B,2
block 3
D,3
C,9
D,1
A,2
B,4
B,5
A,1
C,2
A,x
B,y
B,z

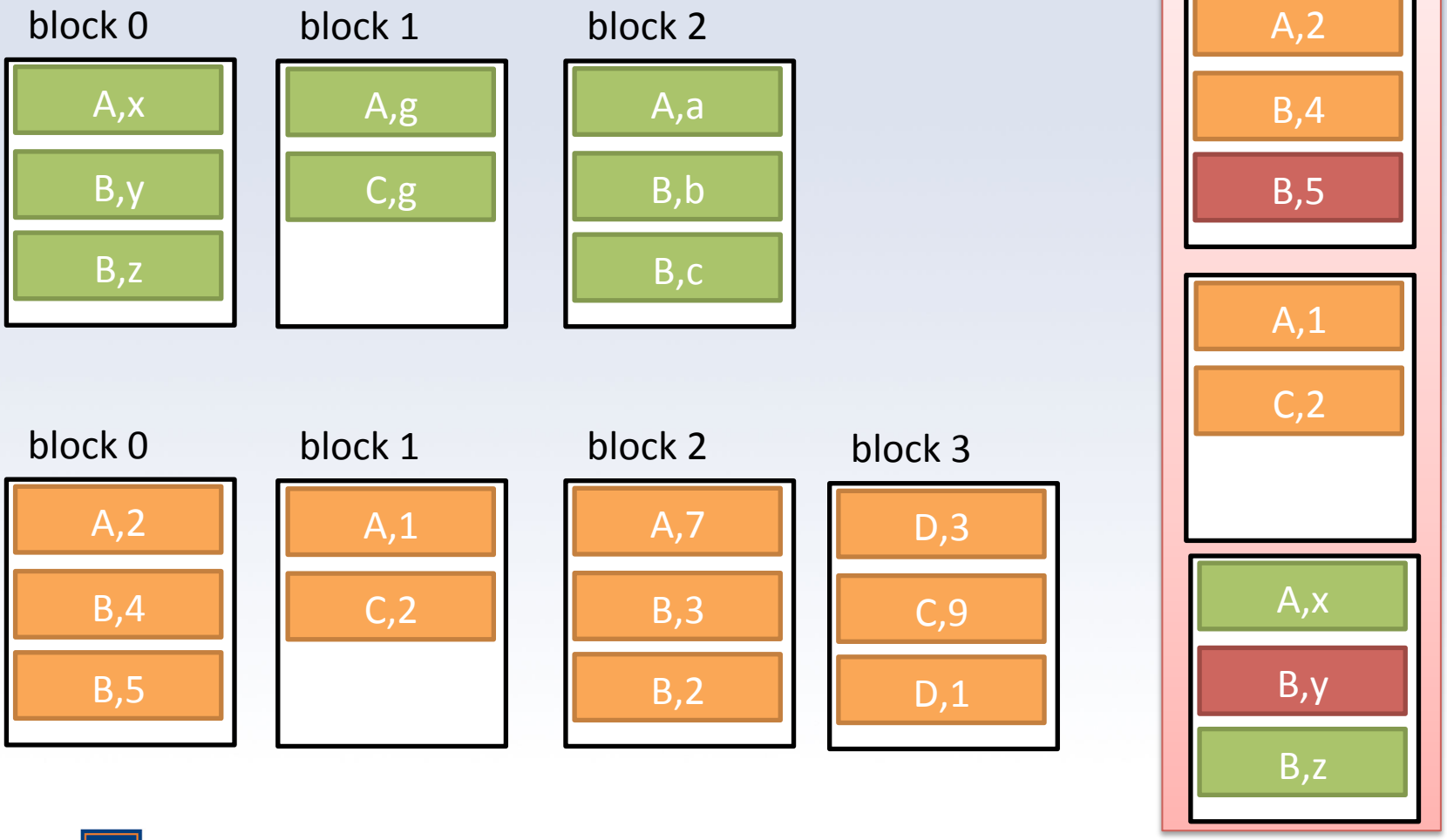
Example: R⋈S


block 0
A,x
B,y
B,z
block 1
A,g
C,g
block 2
A,a
B,b
B,c
block 0
A,2
B,4
B,5
block 1
A,1
C,2
block 2
A,7
B,3
B,2
block 3
D,3
C,9
D,1
A,2
B,4
B,5
A,1
C,2
A,x
B,y
B,z

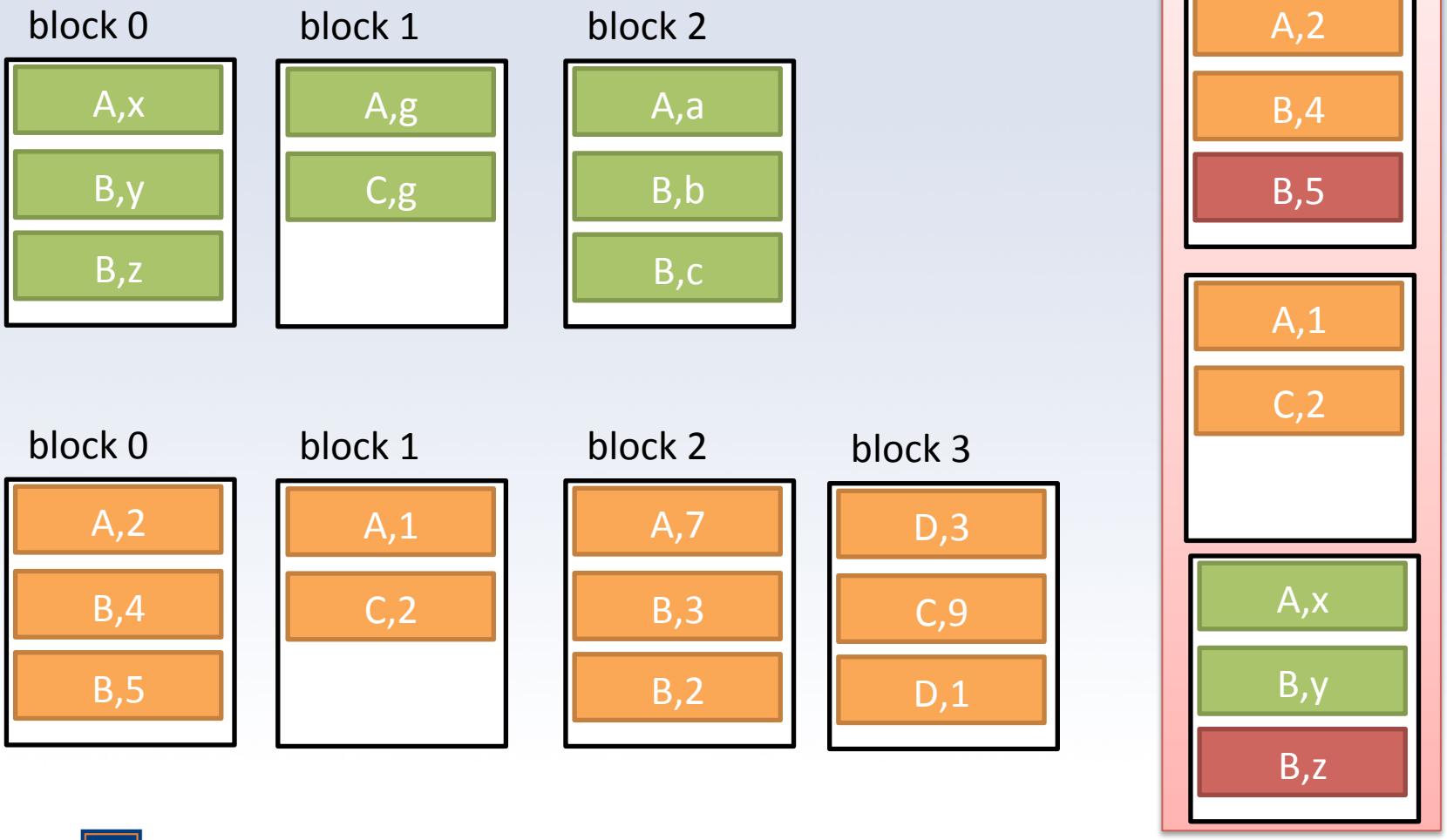
Example: R⋈S


block 0
A,x
B,y
B,z
block 1
A,g
C,g
block 2
A,a
B,b
B,c
block 0
A,2
B,4
B,5
block 1
A,1
C,2
block 2
A,7
B,3
B,2
block 3
D,3
C,9
D,1
A,2
B,4
B,5
A,1
C,2
A,x
B,y
B,z

Example: R⋈S


block 0
A,x
B,y
B,z
block 1
A,g
C,g
block 2
A,a
B,b
B,c
block 0
A,2
B,4
B,5
block 1
A,1
C,2
block 2
A,7
B,3
B,2
block 3
D,3
C,9
D,1
A,2
B,4
B,5
A,1
C,2
A,x
B,y
B,z

Example: R⋈S


block 0
A,x
B,y
B,z
block 1
A,g
C,g
block 2
A,a
B,b
B,c
block 0
A,2
B,4
B,5
block 1
A,1
C,2
block 2
A,7
B,3
B,2
block 3
D,3
C,9
D,1
A,2
B,4
B,5
A,1
C,2
A,x
B,y
B,z

Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

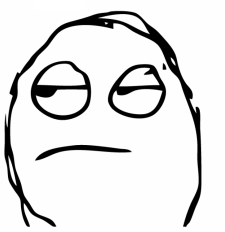
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,x
B,y
B,z



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

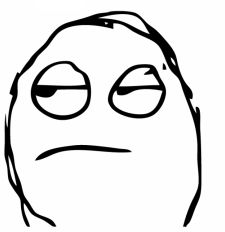
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,x
B,y
B,z



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

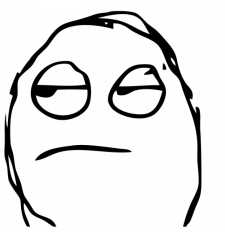
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,x
B,y
B,z



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

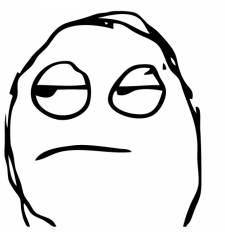
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,x
B,y
B,z



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

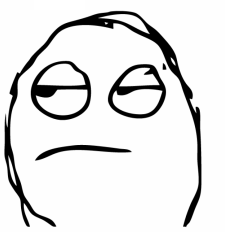
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,x
B,y
B,z



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

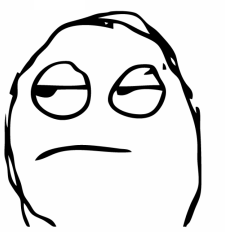
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,x
B,y
B,z



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

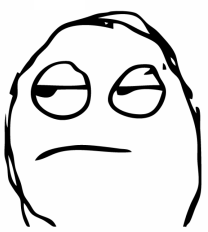
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,g
C,g



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

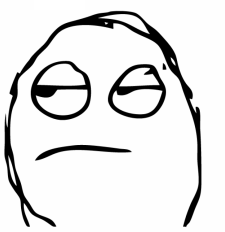
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,g
C,g



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

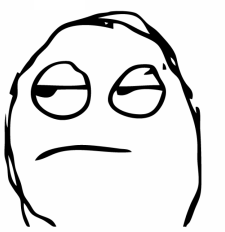
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,g
C,g



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

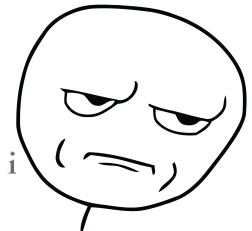
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,g
C,g



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

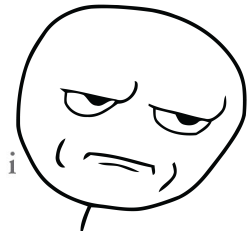
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,g
C,g



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

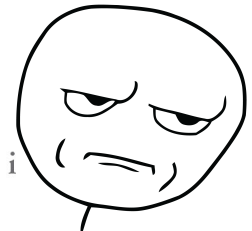
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,g
C,g



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

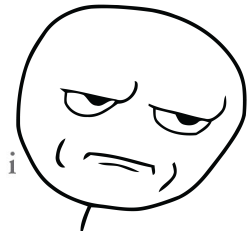
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,g
C,g



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

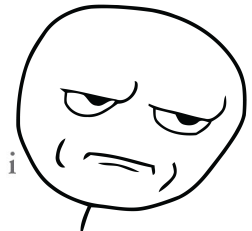
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,g
C,g



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

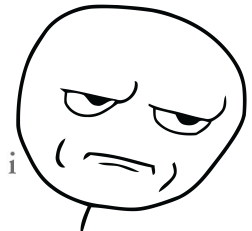
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,g
C,g



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

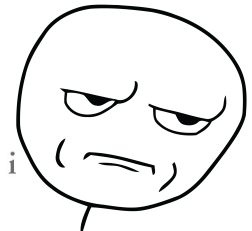
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,g
C,g



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

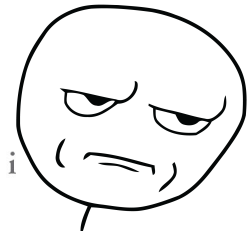
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,g
C,g



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

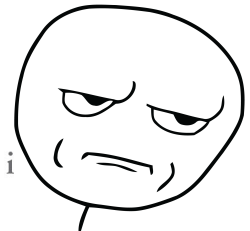
A,7
B,3
B,2

block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

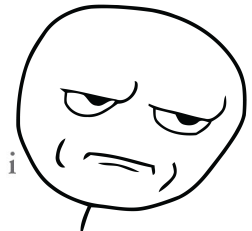
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,a
B,b
B,c



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

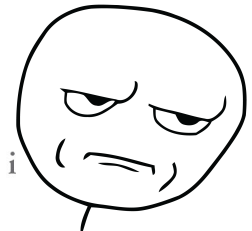
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,a
B,b
B,c



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

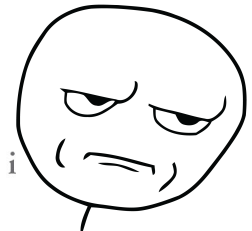
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,a
B,b
B,c



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

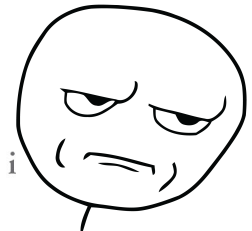
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,a
B,b
B,c



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

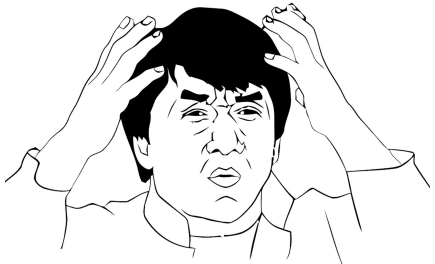
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,a
B,b
B,c



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

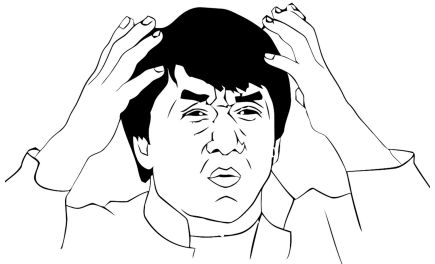
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,a
B,b
B,c



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

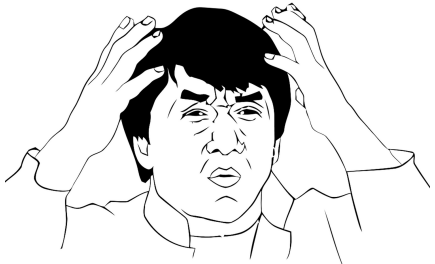
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,a
B,b
B,c



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

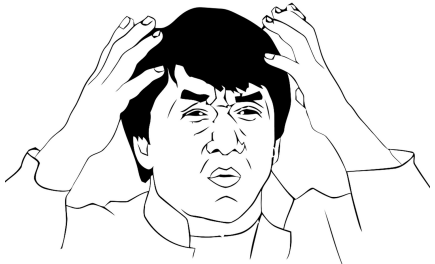
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,a
B,b
B,c



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

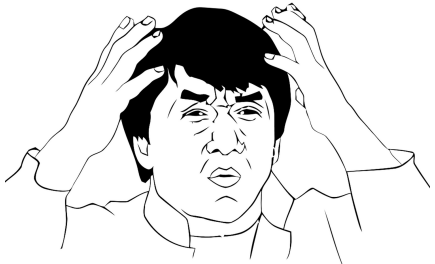
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,a
B,b
B,c



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

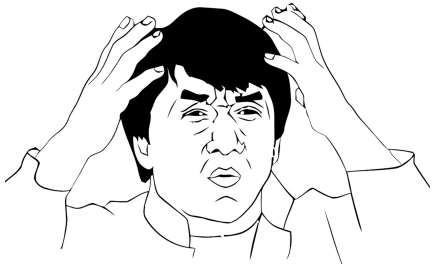
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,a
B,b
B,c



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

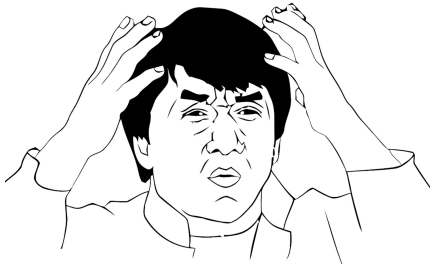
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,a
B,b
B,c



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

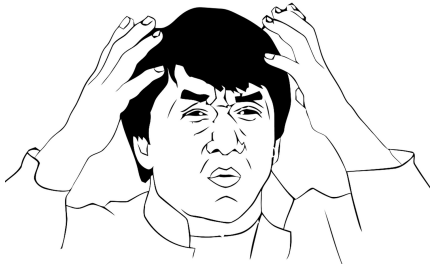
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,a
B,b
B,c



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

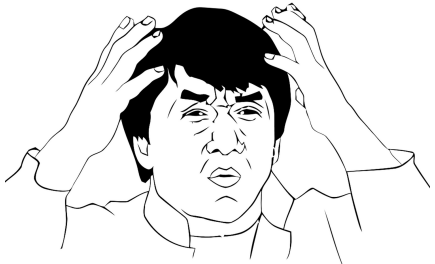
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,a
B,b
B,c



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

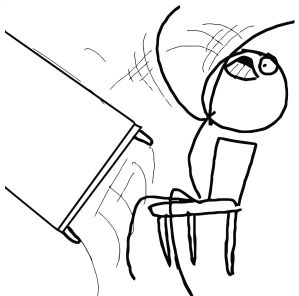
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,a
B,b
B,c



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

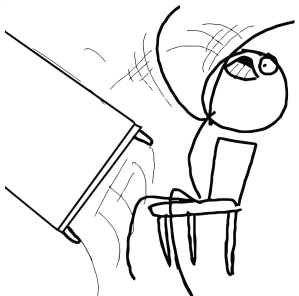
block 2

A,7
B,3
B,2

block 3

D,3
C,9
D,1

A,2
B,4
B,5
A,1
C,2
A,a
B,b
B,c



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

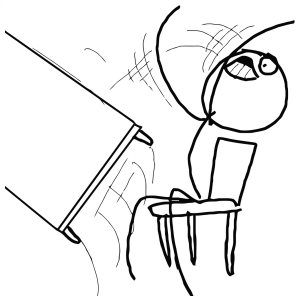
block 3

D,3
C,9
D,1

A,2
B,4
B,5

A,1
C,2

A,a
B,b
B,c



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

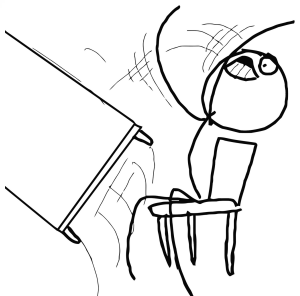
A,1
C,2

block 2

A,7
B,3
B,2

block 3

D,3
C,9
D,1



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

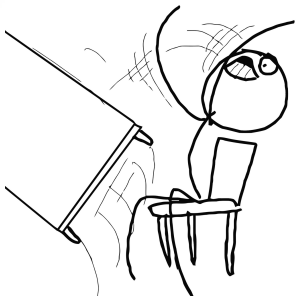
A,7
B,3
B,2

block 3

D,3
C,9
D,1

A,7
B,3
B,2

D,3
C,9
D,1



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

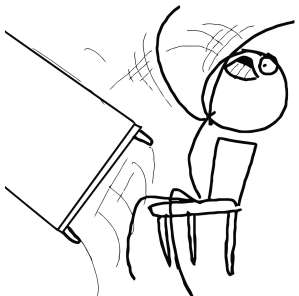
block 2

A,7
B,3
B,2

block 3

D,3
C,9
D,1

A,7
B,3
B,2
D,3
C,9
D,1
A,x
B,y
B,z



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

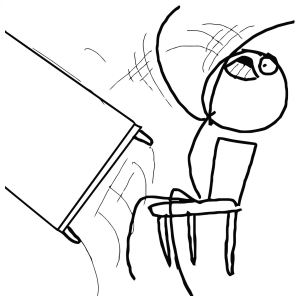
block 3

D,3
C,9
D,1

A,7
B,3
B,2

D,3
C,9
D,1

A,x
B,y
B,z



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

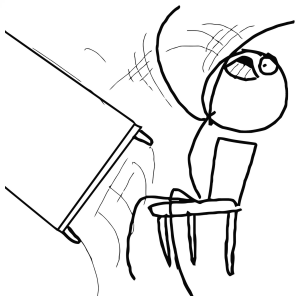
block 3

D,3
C,9
D,1

A,7
B,3
B,2

D,3
C,9
D,1

A,x
B,y
B,z



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

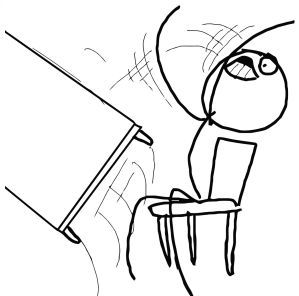
block 2

A,7
B,3
B,2

block 3

D,3
C,9
D,1

A,7
B,3
B,2
D,3
C,9
D,1
A,x
B,y
B,z



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

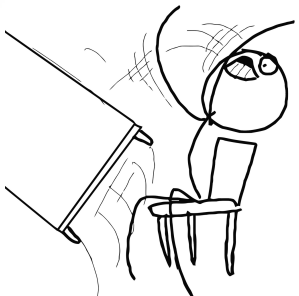
block 2

A,7
B,3
B,2

block 3

D,3
C,9
D,1

A,7
B,3
B,2
D,3
C,9
D,1
A,x
B,y
B,z



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

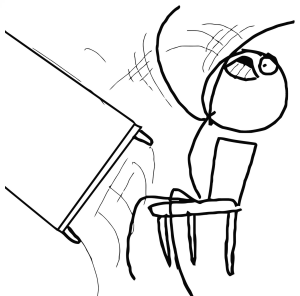
block 3

D,3
C,9
D,1

A,7
B,3
B,2

D,3
C,9
D,1

A,x
B,y
B,z



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

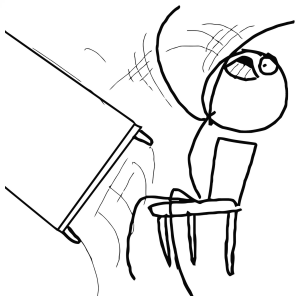
block 2

A,7
B,3
B,2

block 3

D,3
C,9
D,1

A,7
B,3
B,2
D,3
C,9
D,1
A,x
B,y
B,z



Example: R⋈S

block 0

A,x
B,y
B,z

block 1

A,g
C,g

block 2

A,a
B,b
B,c

block 0

A,2
B,4
B,5

block 1

A,1
C,2

block 2

A,7
B,3
B,2

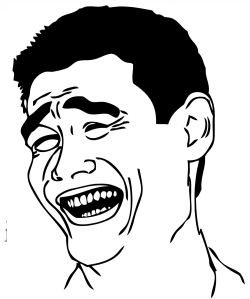
block 3

D,3
C,9
D,1

A,7
B,3
B,2

D,3
C,9
D,1

A,x
B,y
B,z



Pseudocode: Block-based

```
for (M-1) blocks bs in S:
  for block br in R:
    for s in bs:
      for r in br:
        t=join(r,s)
        if(t):
          output t
```

Cost of NBLJ

- Read S once (outer loop)
  - B(S)/(M-1) iterations
  - Reading *all* of R each time
- Total cost: B(S)B(R)/(M-1)
- Note: iterate over smaller relation first
  - keeps B(S)/(M-1) small
  - fewer reads of R

One-Pass/NBLJ Summary

Operator	M required	I/O Cost
σ, π	1	B
δ, γ	B	B
$\cup, \cap, -, \bowtie, \times$	min(B(R),B(S))	B(R)+B(S)
$\bowtie$	$M \geq 2$	B(R)B(S)/M

Sort-Based Algorithms

- Can base implementation of operations on merge sort
  - Special implementation for large data
  - Called *Two-Phase Multiway Merge-Sort*
  - TPMMS

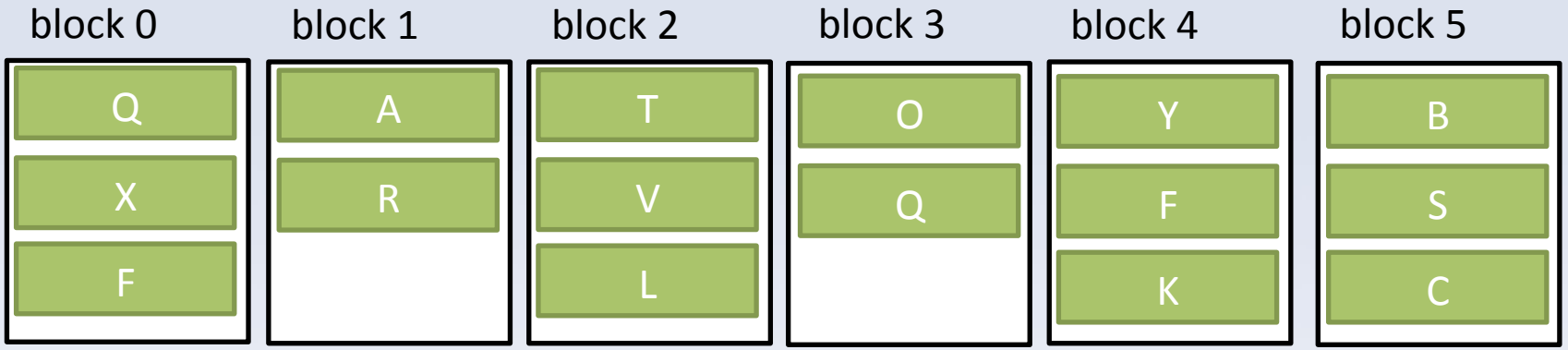
TPMMS

- Has two phases
  - Phase 1: sort memory sized sublists
  - Phase 2: merge sorted sublists
- Requires $\sqrt{B(R)} \leq M$

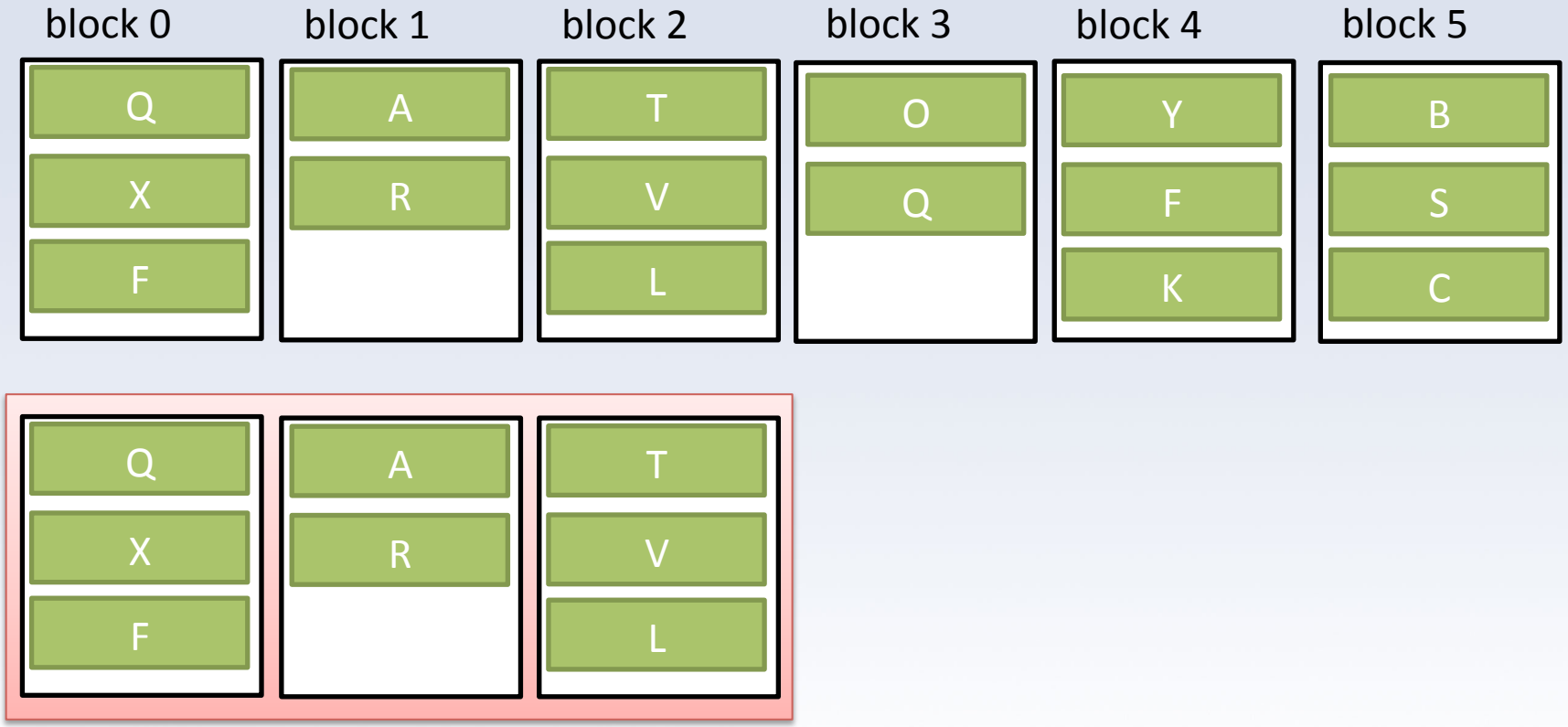
TPMMS

- Phase 1: Repeat until R is exhausted:
  - Fill up *all* M buffers with blocks from R
  - Sort using quicksort
  - Write resulting sublist to disk
- Phase 2: Repeat until sublists exhausted:
  - Load buffers with smallest block for each sublist
  - Identify smallest elements, move to output block
  - When output block full, write to disk
  - If list buffer is empty, get next one from disk

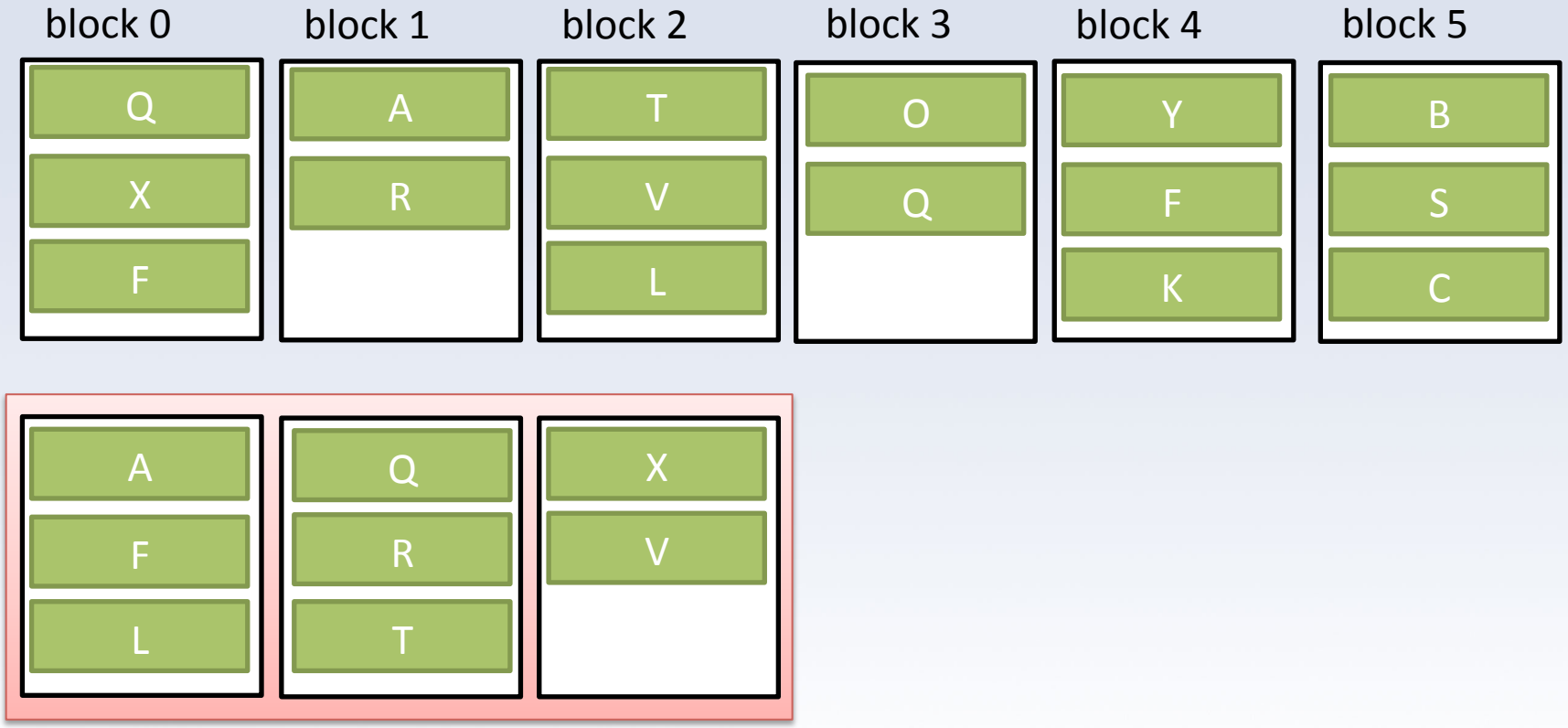
Example


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

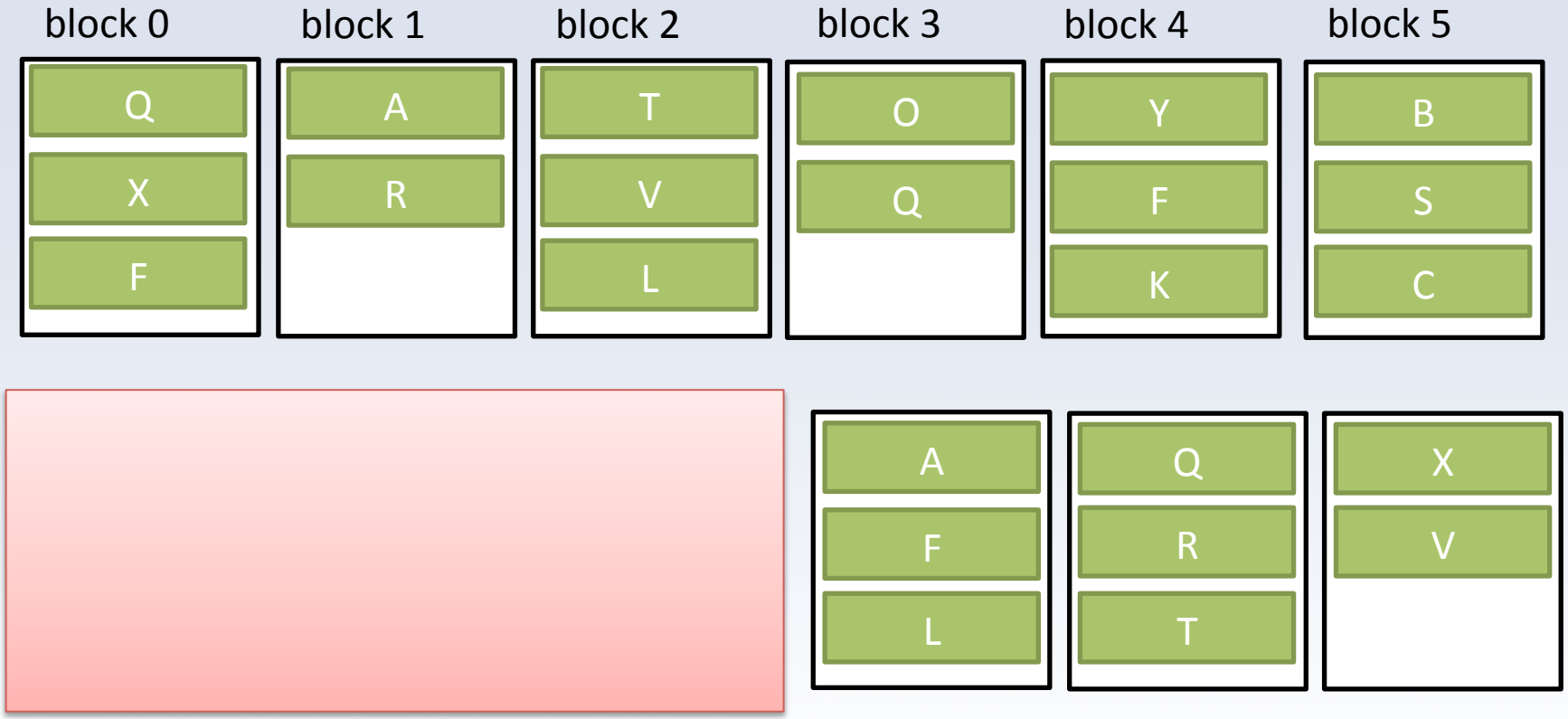
Phase 1


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

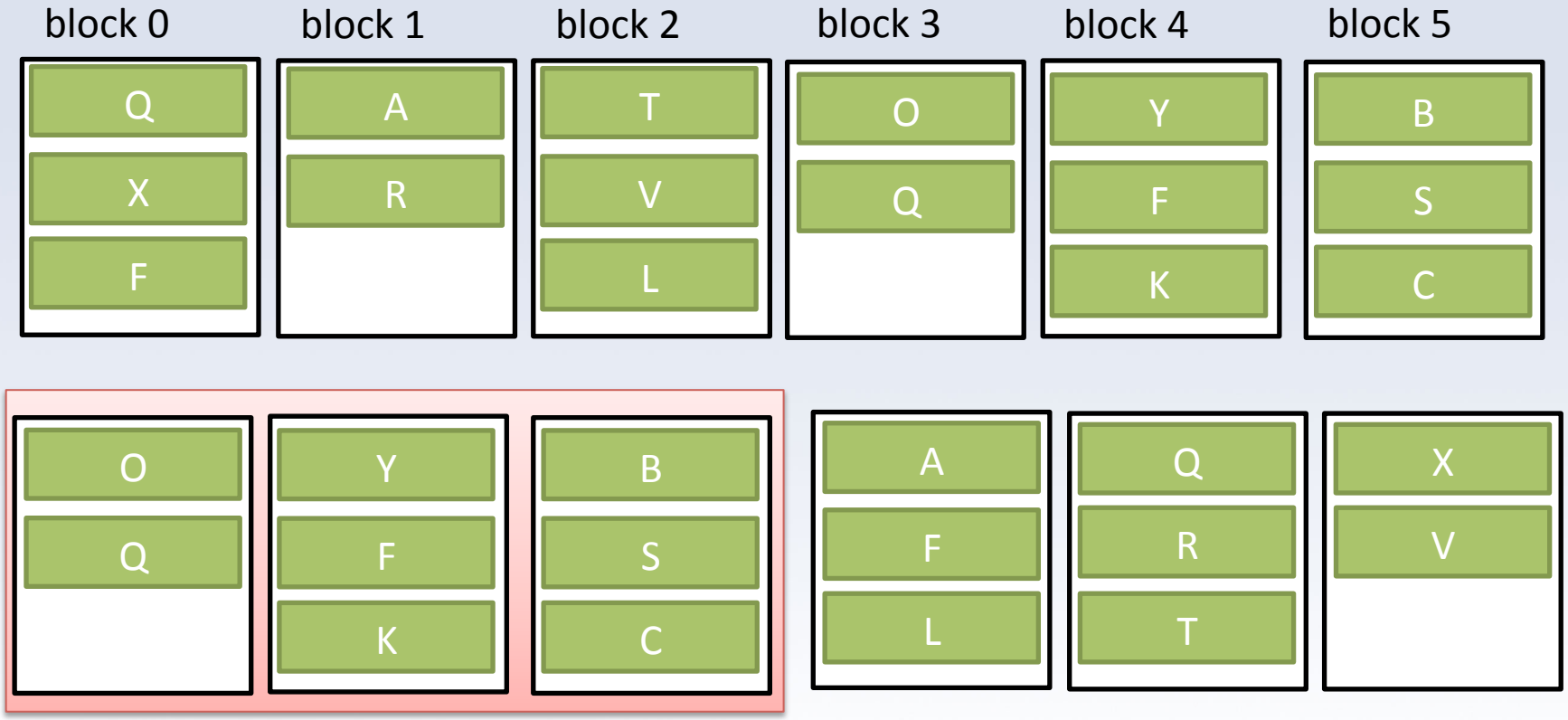
Phase 1


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
X
V

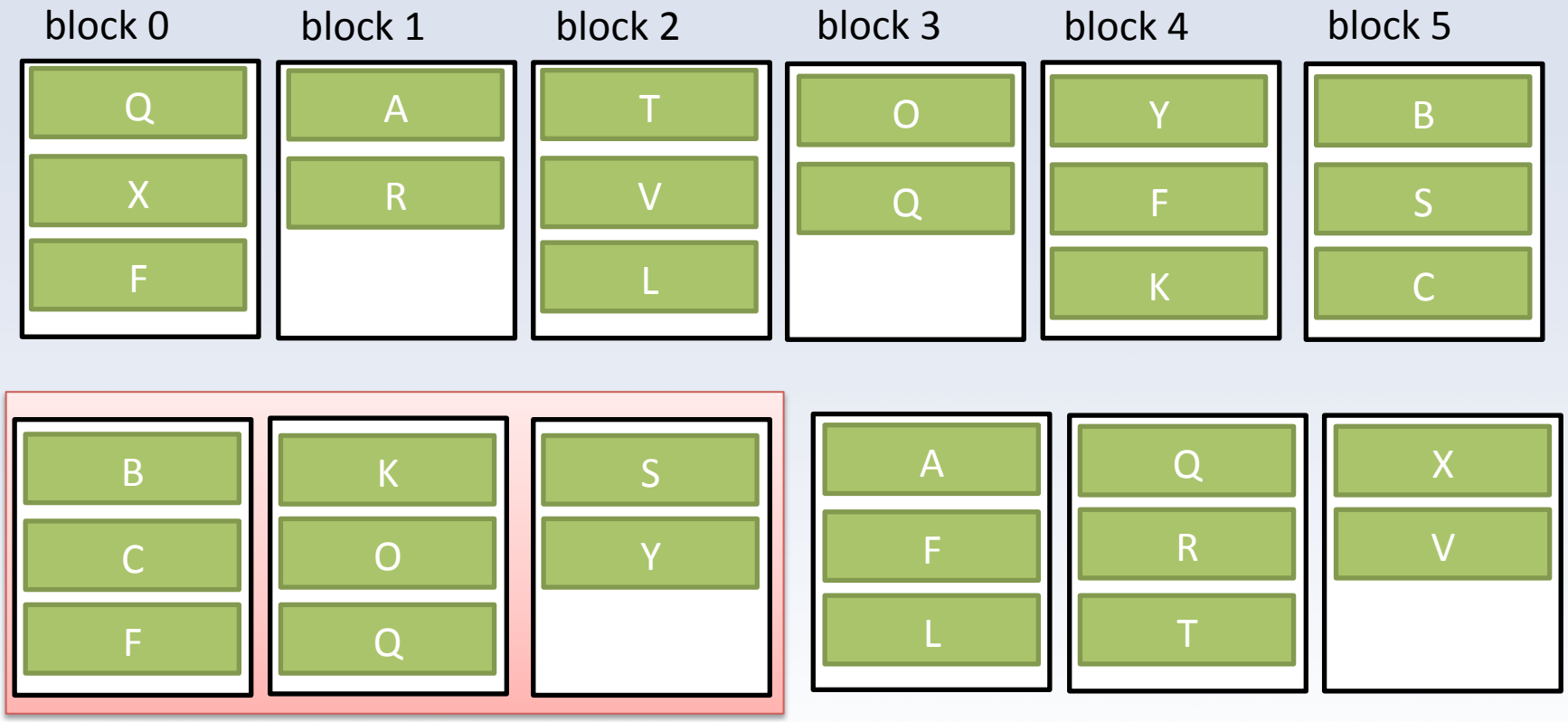
Phase 1


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
X
V

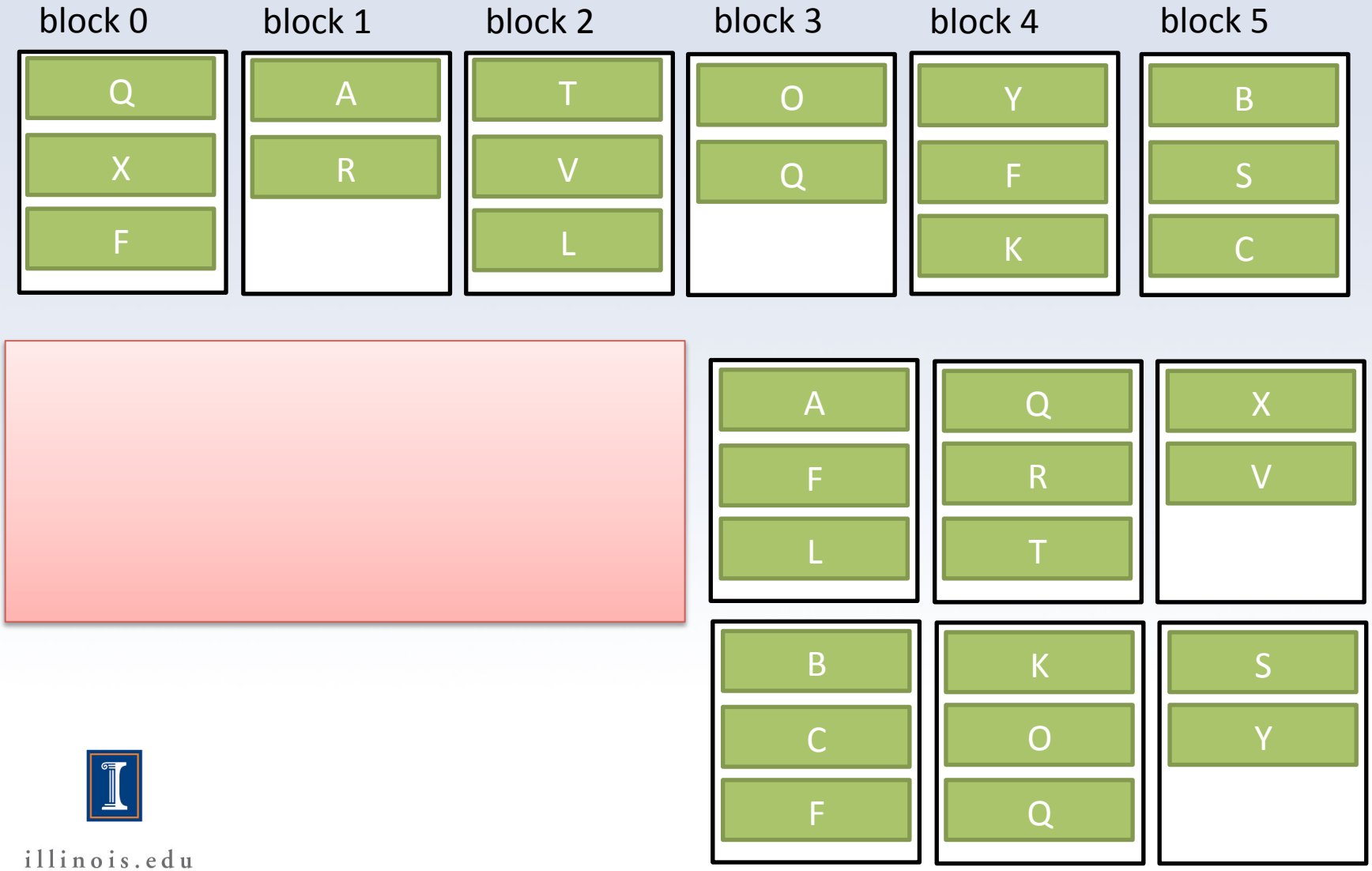
Phase 1


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
O
Q
Y
F
K
B
S
C
A
F
L
Q
R
T
X
V

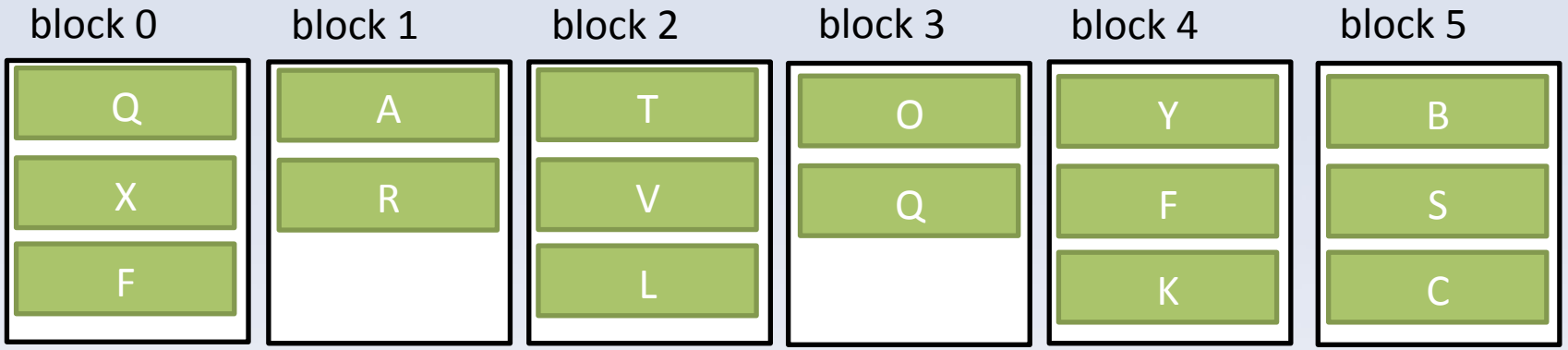
Phase 1

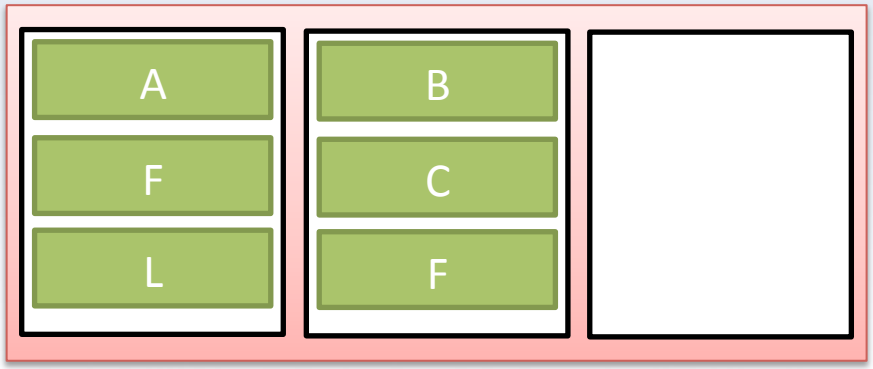

block 0
block 1
block 2
block 3
block 4
block 5
Q
X
F
A
R
T
V
L
O
Q
Y
F
K
B
S
C
B
C
F
K
O
Q
S
Y
A
F
L
Q
R
T
X
V

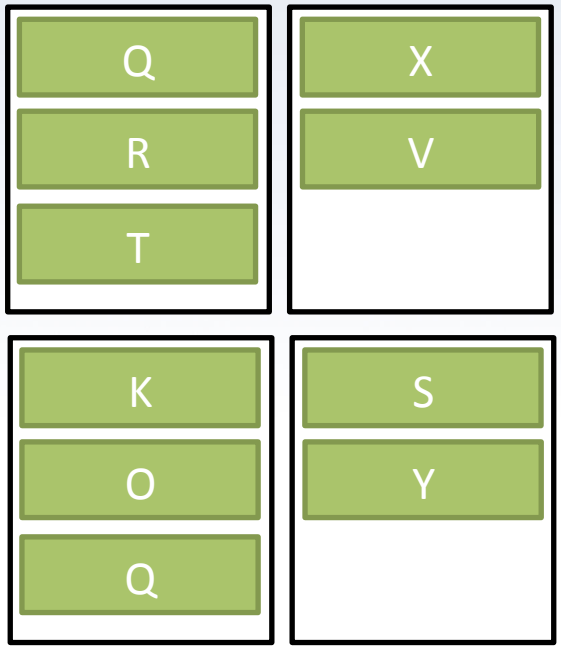
Phase 1


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
X
V
B
C
F
K
O
Q
S
Y

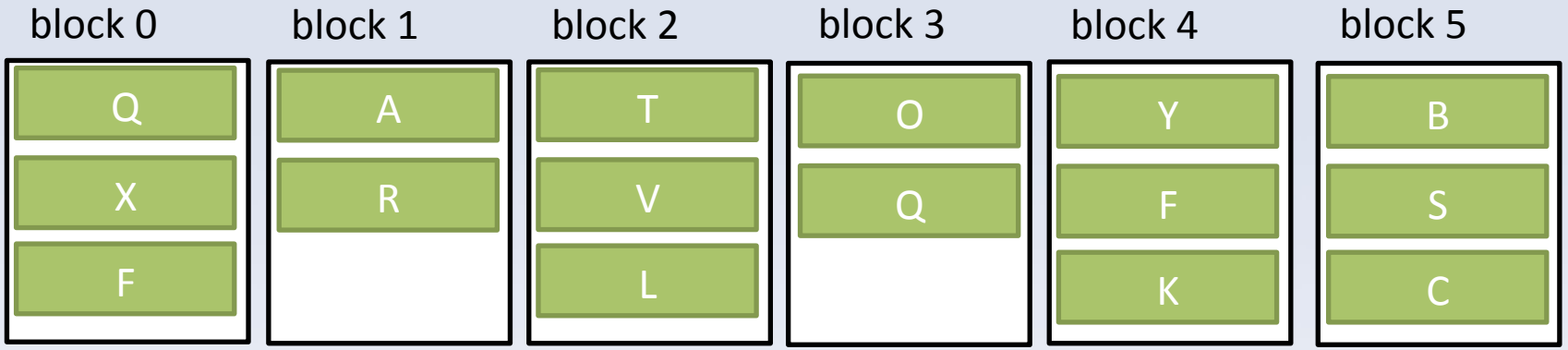
Phase 2

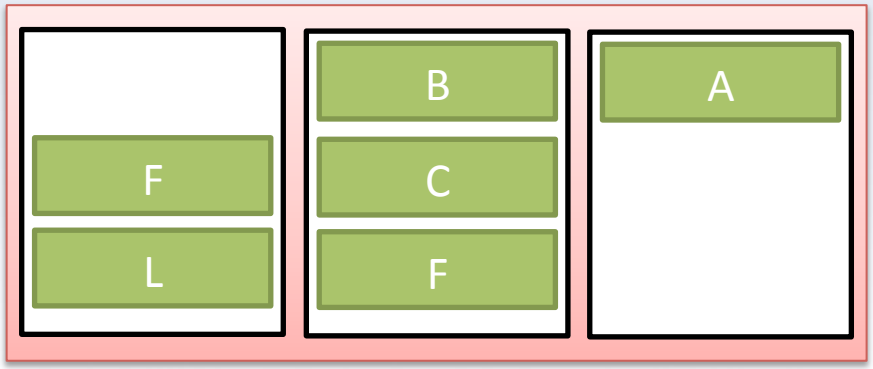

block 0
block 1
block 2
block 3
block 4
block 5
Q
X
F
A
R
T
V
L
O
Q
Y
F
K
B
S
C

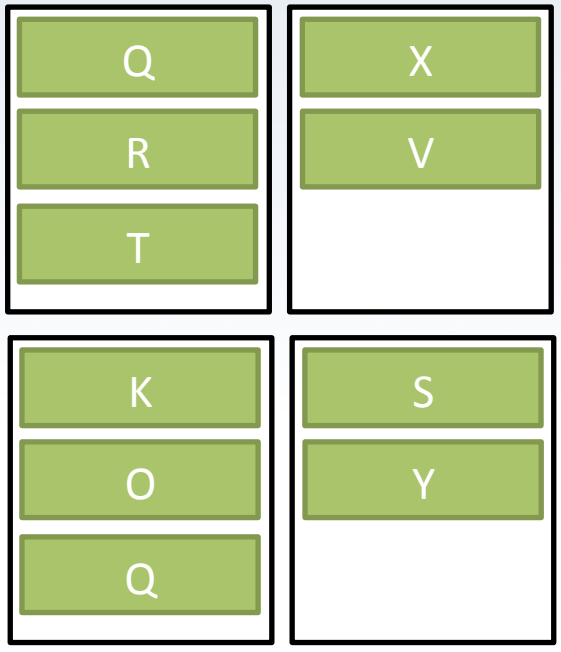

A
F
L
B
C
F


Q
R
T
X
V
K
O
Q
S
Y

Phase 2

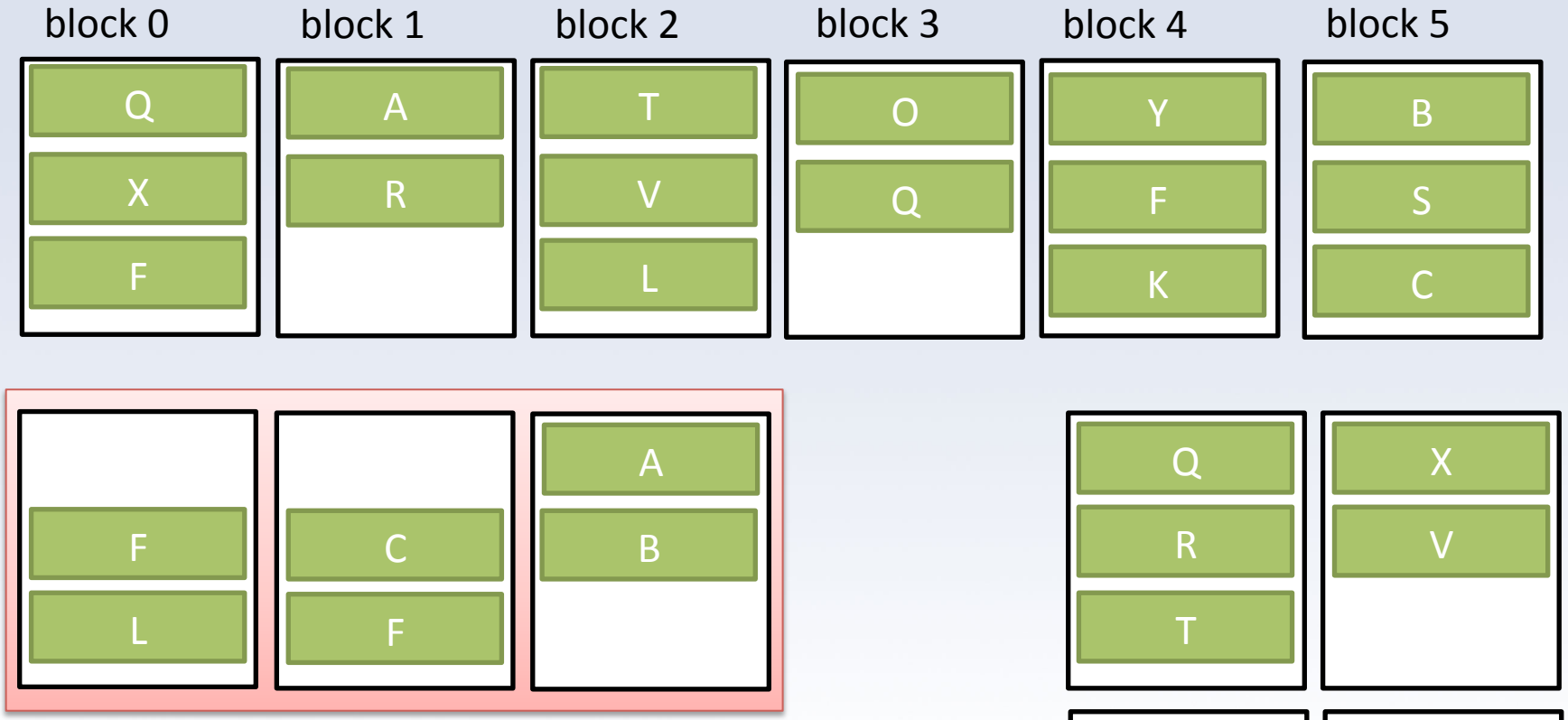






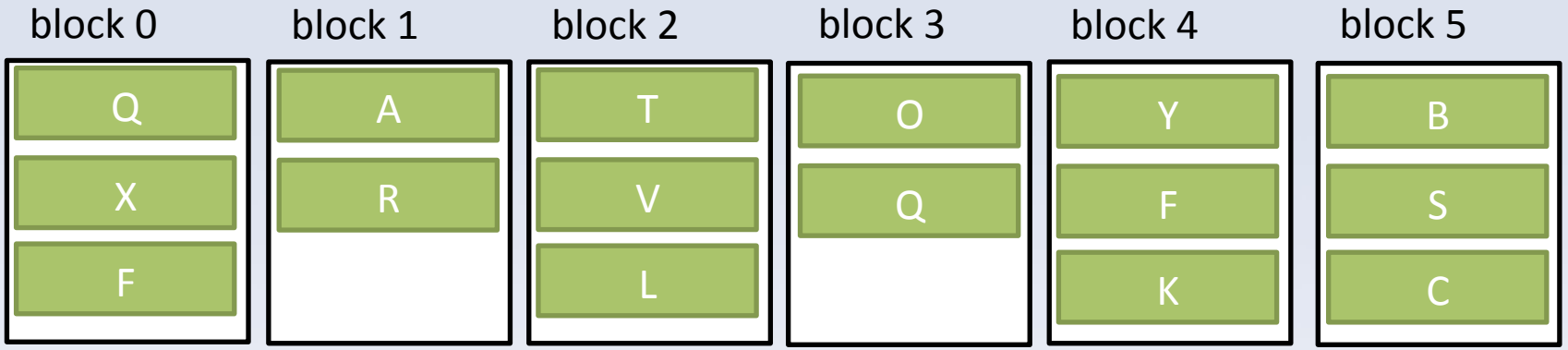
illinois.edu

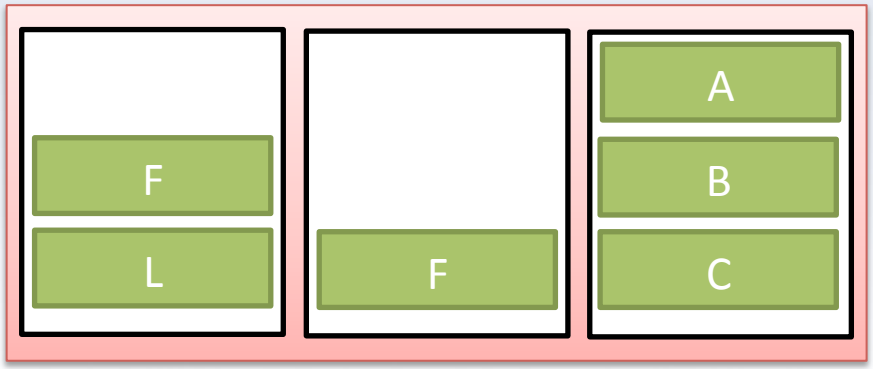
Phase 2

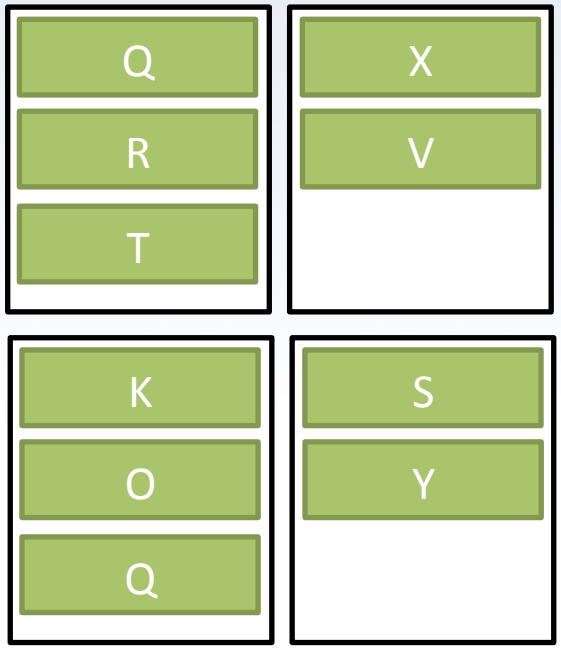


illinois.edu

Phase 2

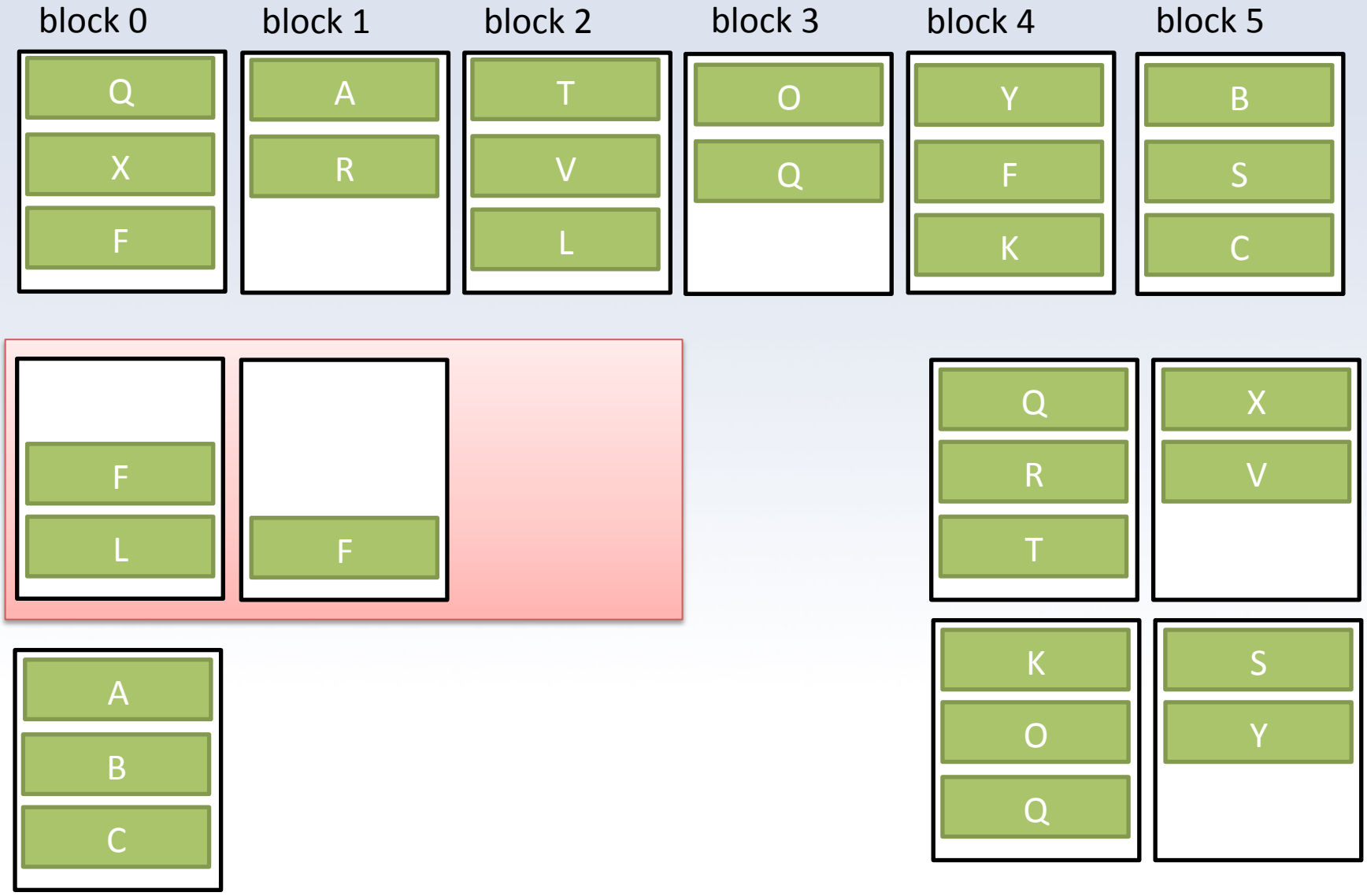




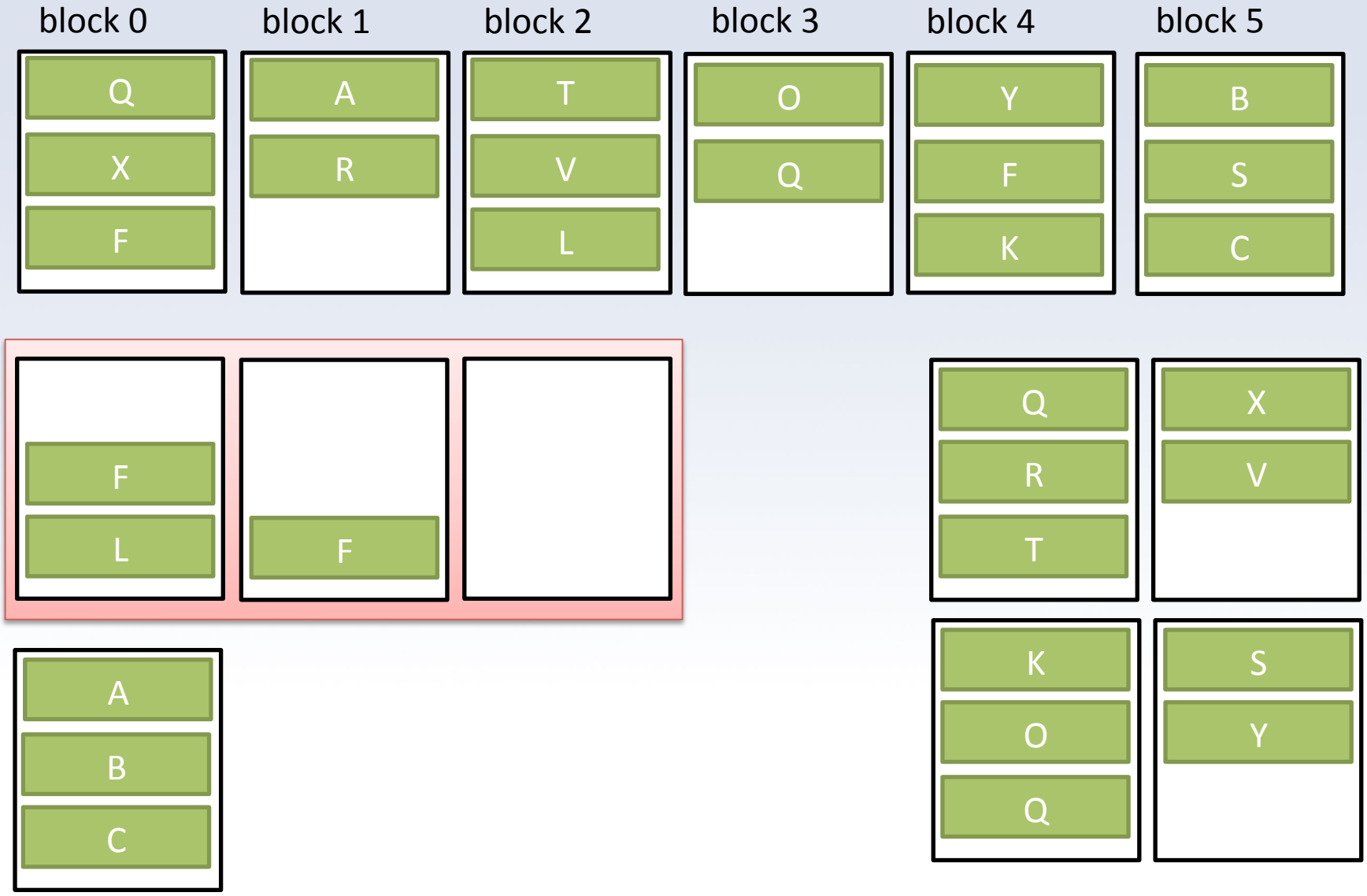


illinois.edu

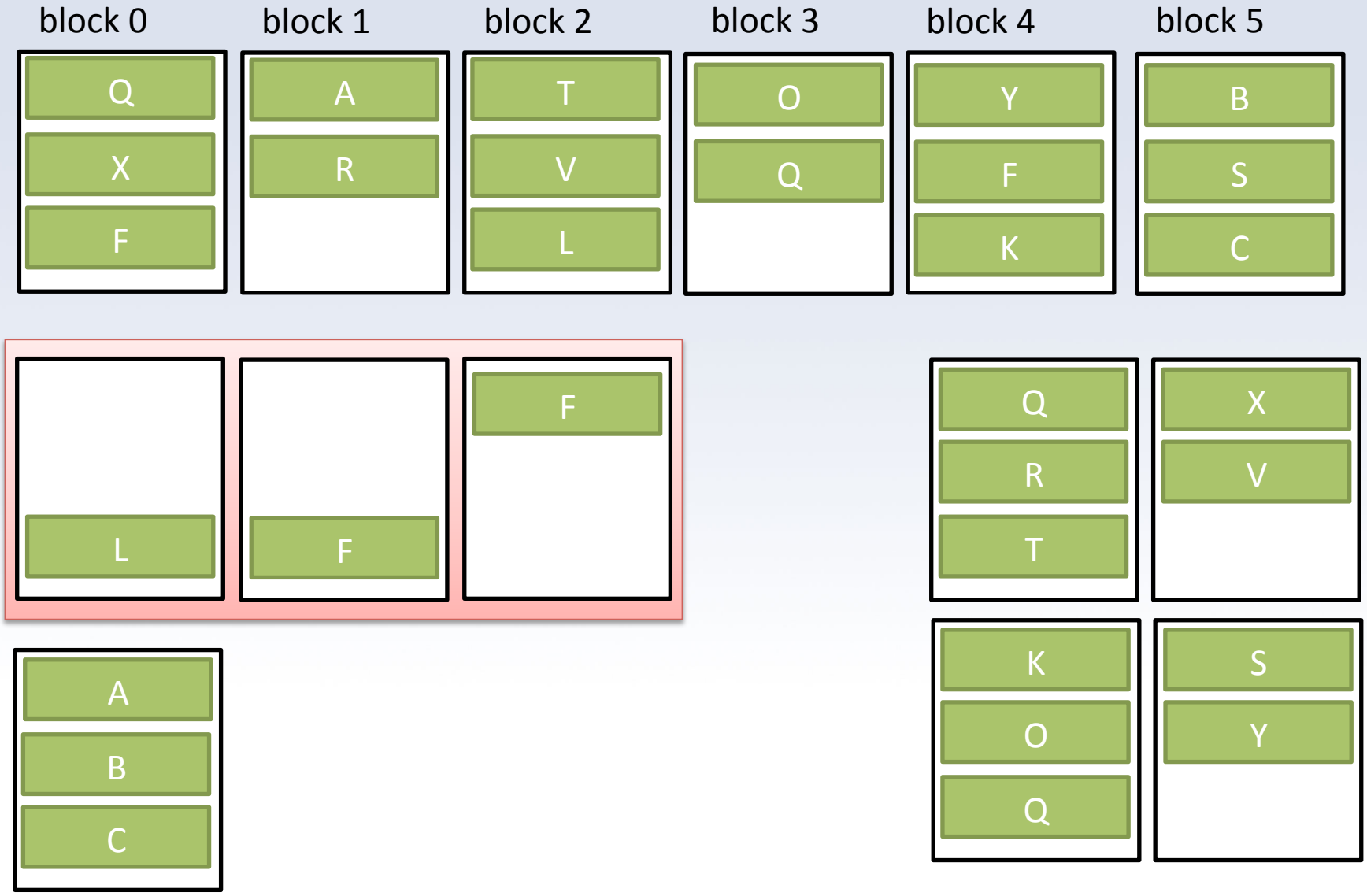
Phase 2


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
X
V
K
O
Q
S
Y
A
B
C

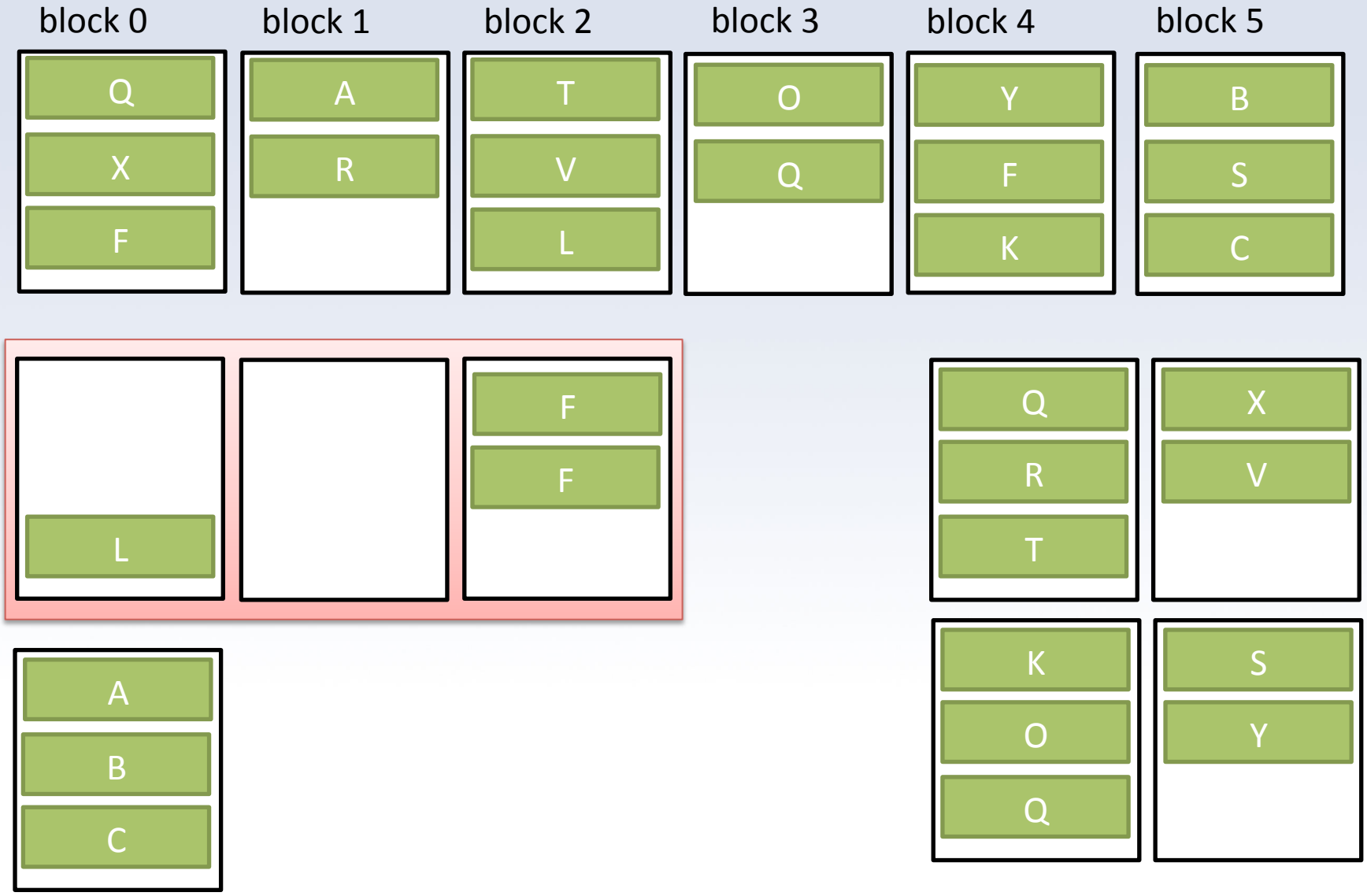
Phase 2


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
X
V
K
O
Q
S
Y
A
B
C

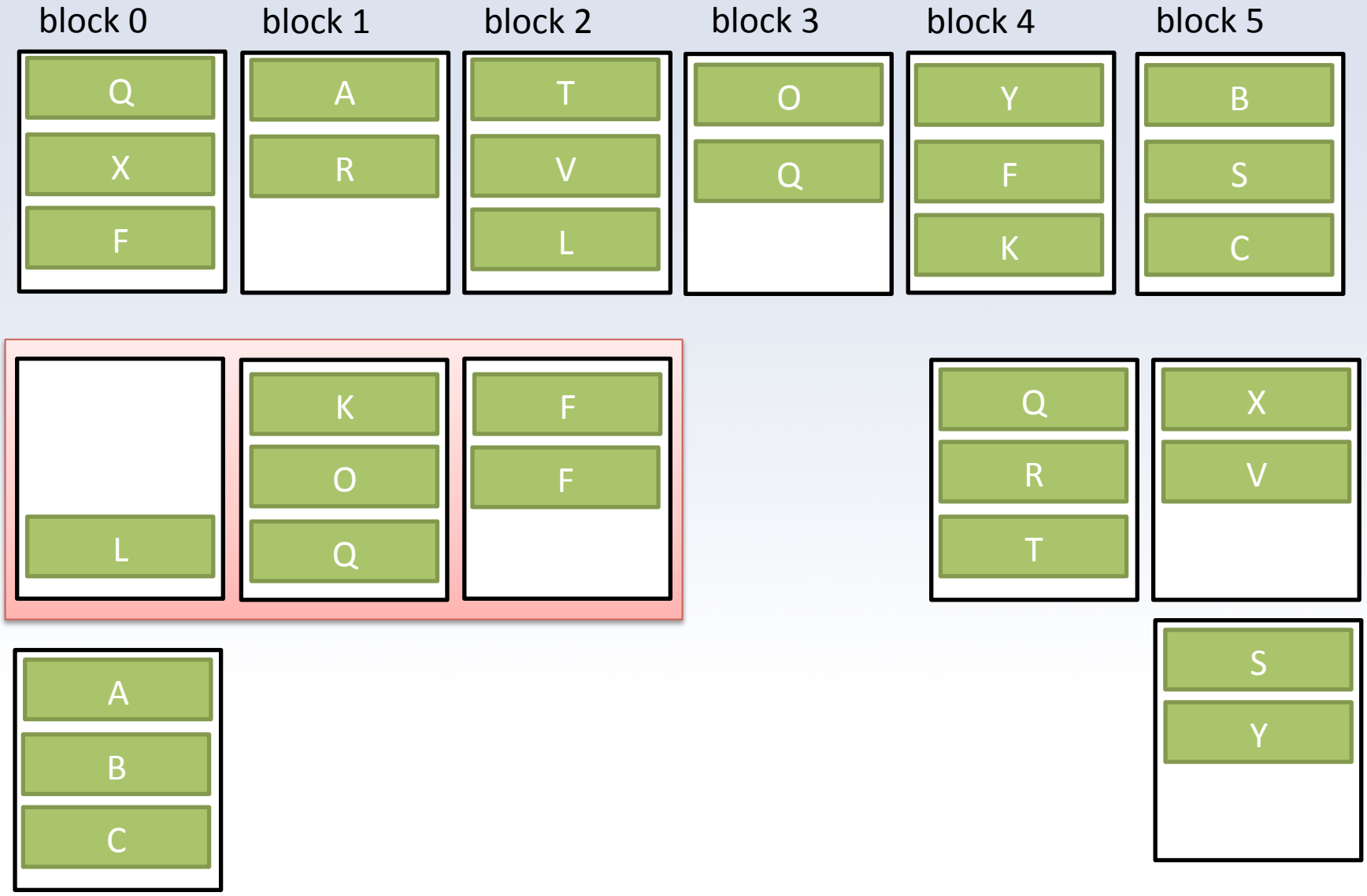
Phase 2


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
X
V
K
O
Q
S
Y
A
B
C

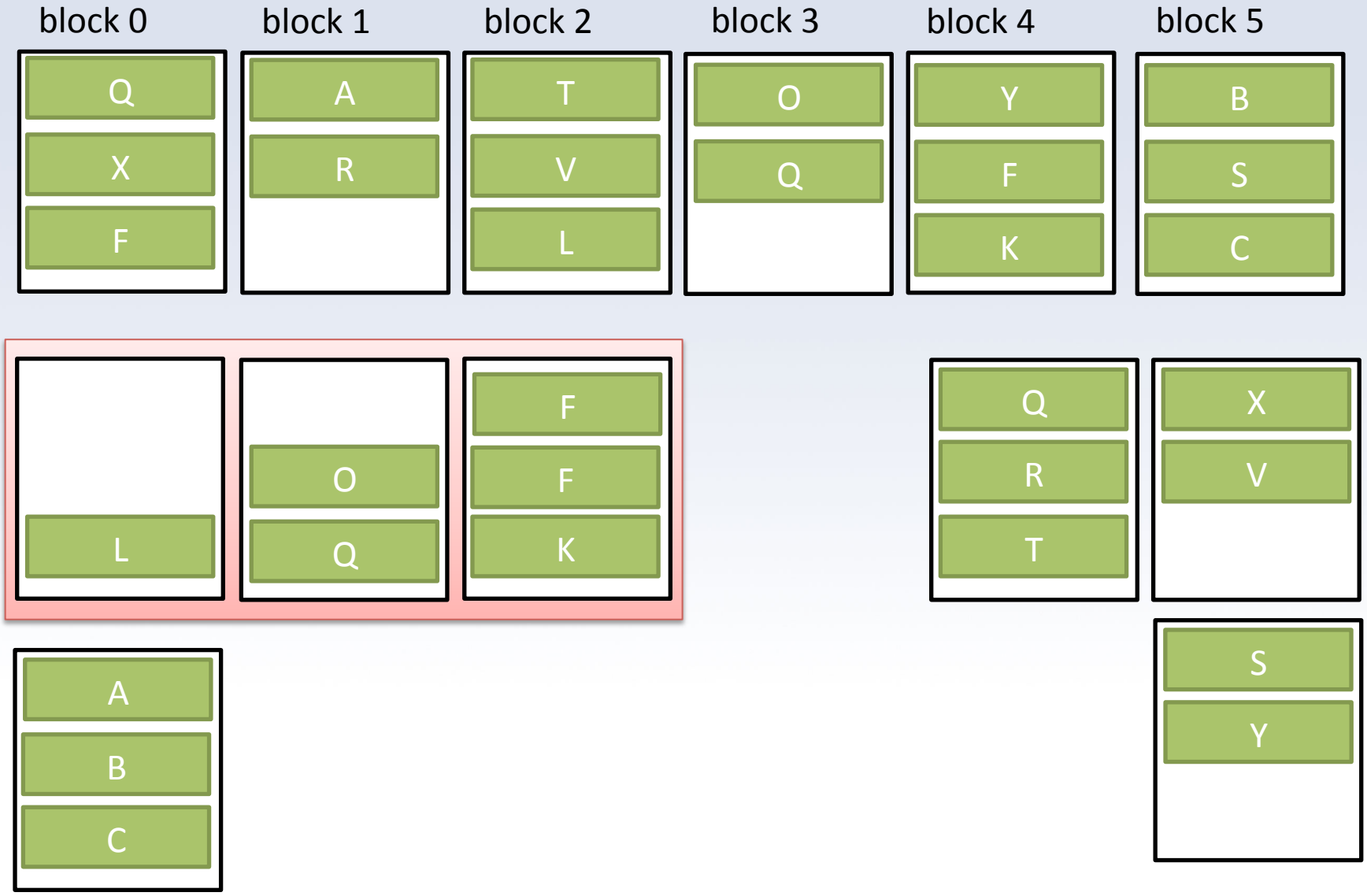
Phase 2


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
X
V
A
B
C
K
O
Q
S
Y

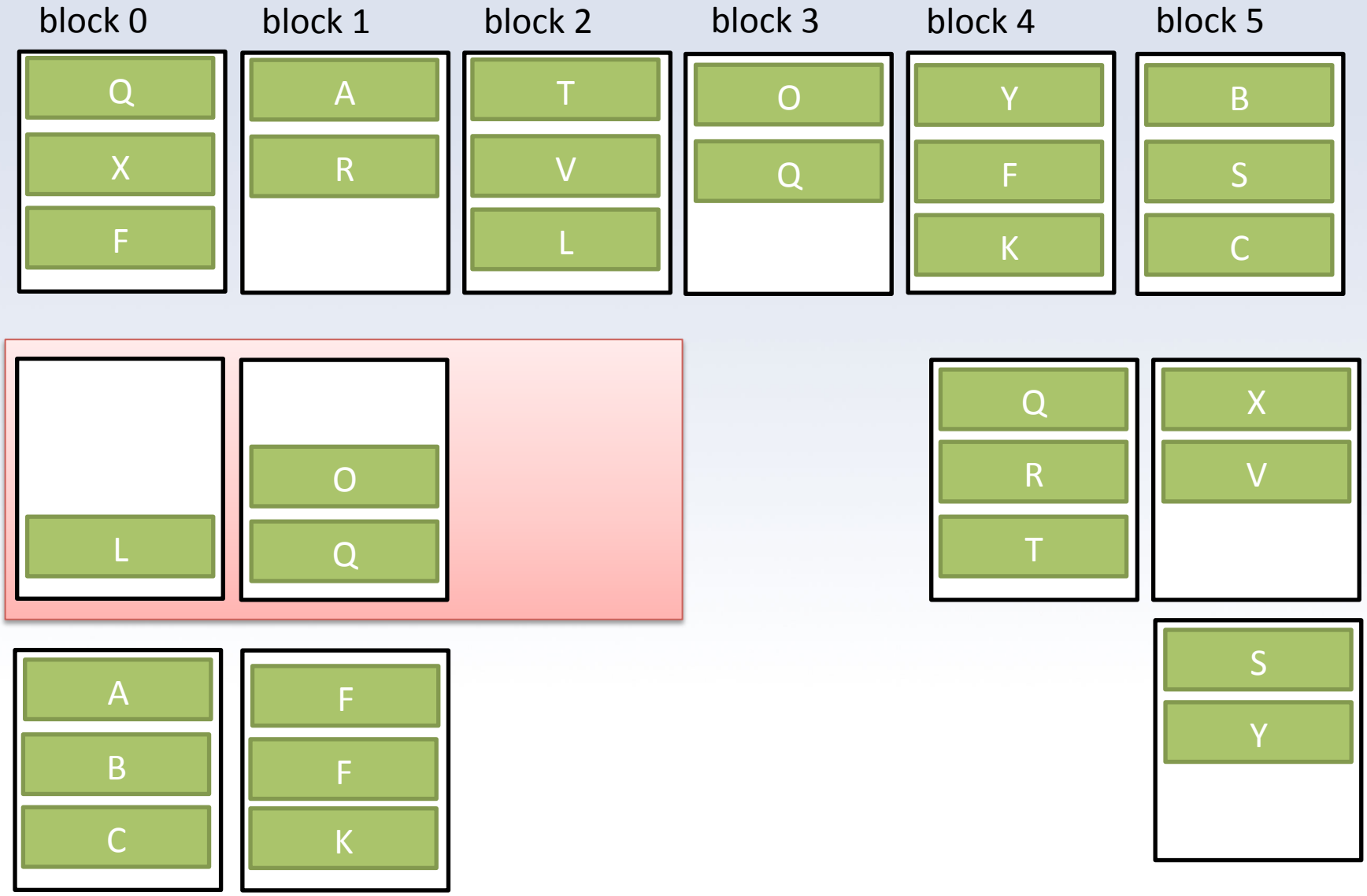
Phase 2


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
F
Q
R
T
X
V
A
B
C
S
Y

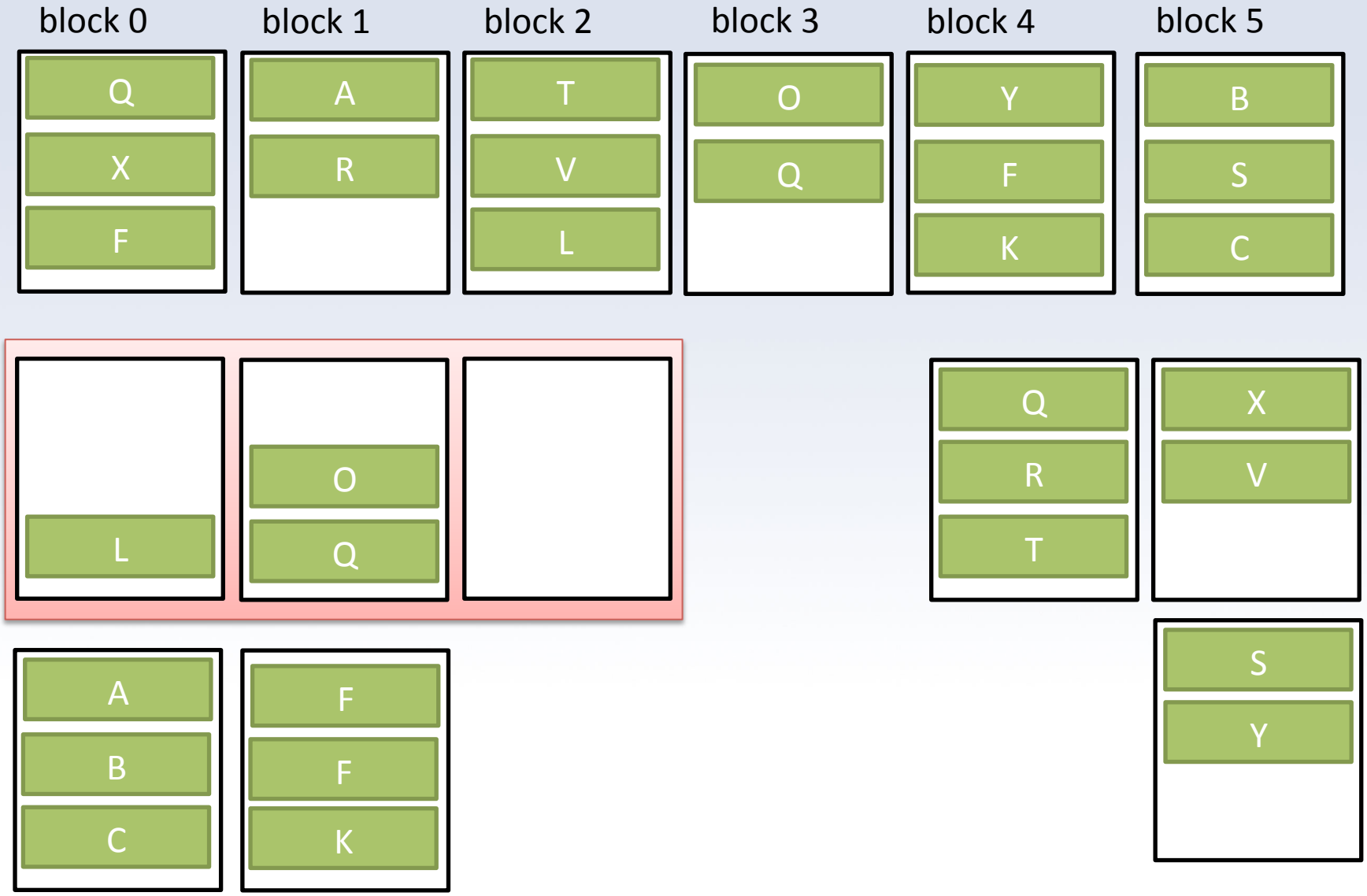
Phase 2


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
F
K
Q
R
T
X
V
A
B
C
S
Y

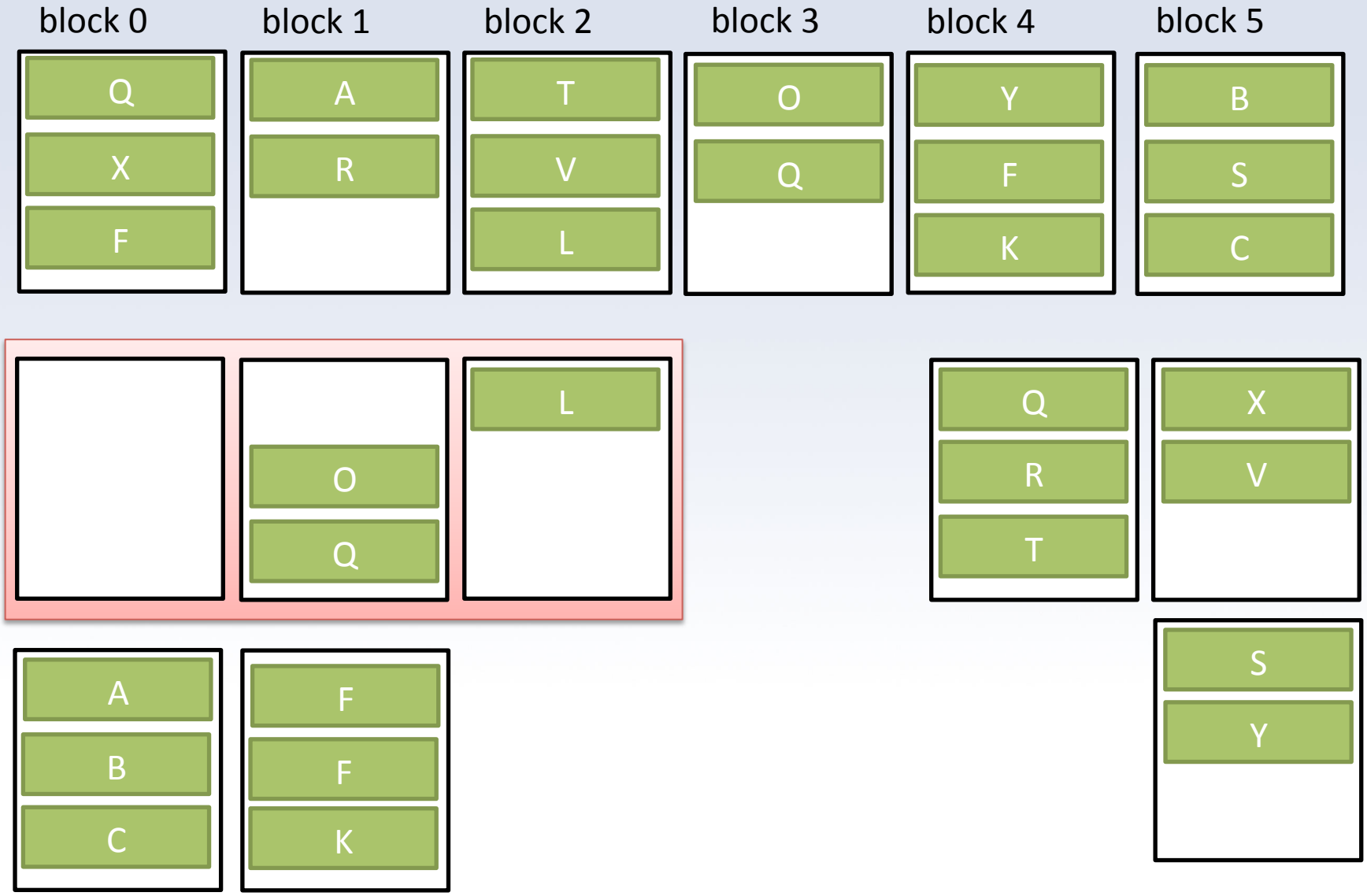
Phase 2


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
L
O
Q
Q
R
T
X
V
A
B
C
F
F
K
S
Y

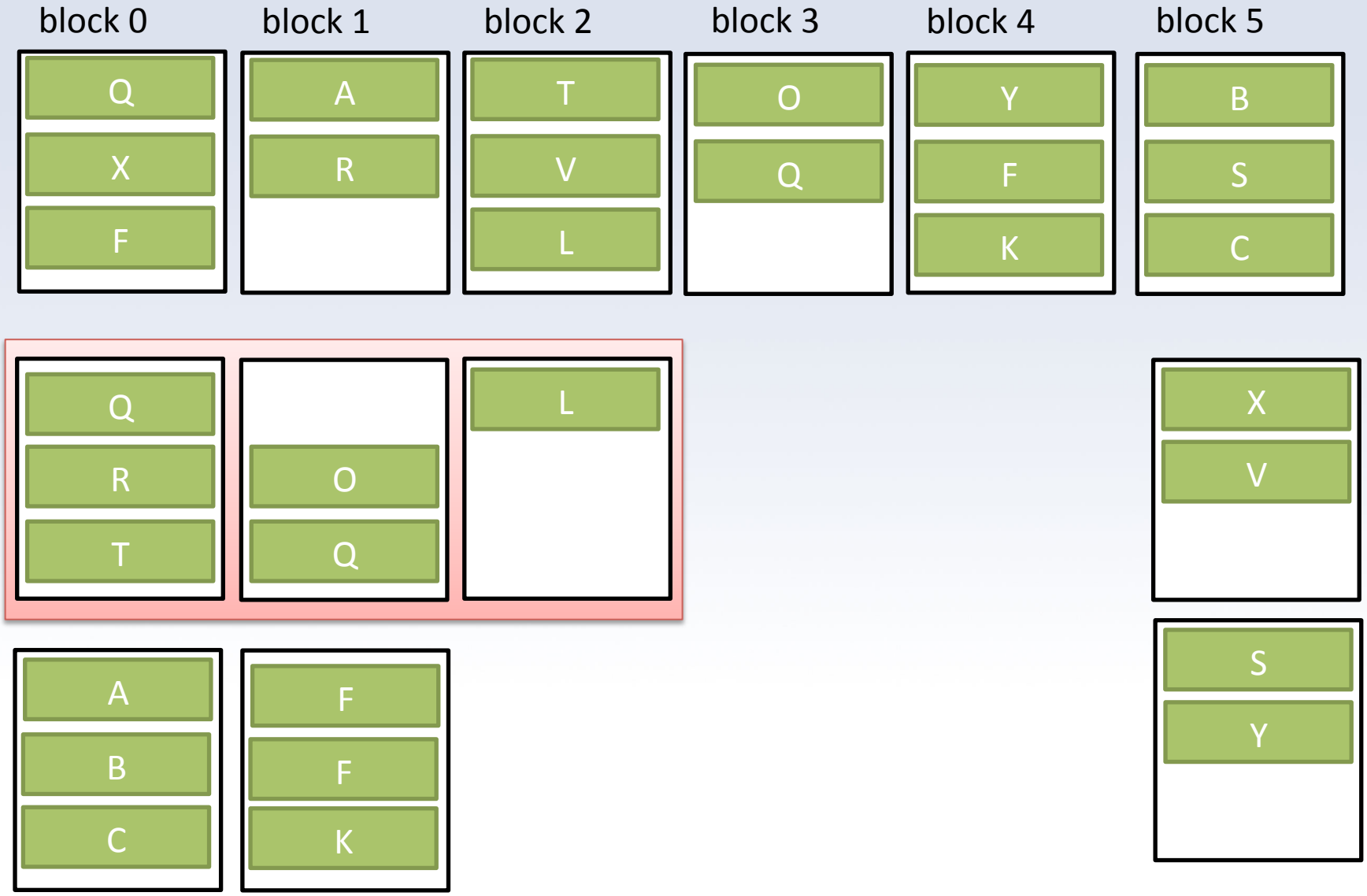
Phase 2


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
Q
R
T
X
V
S
Y
A
B
C
F
F
K

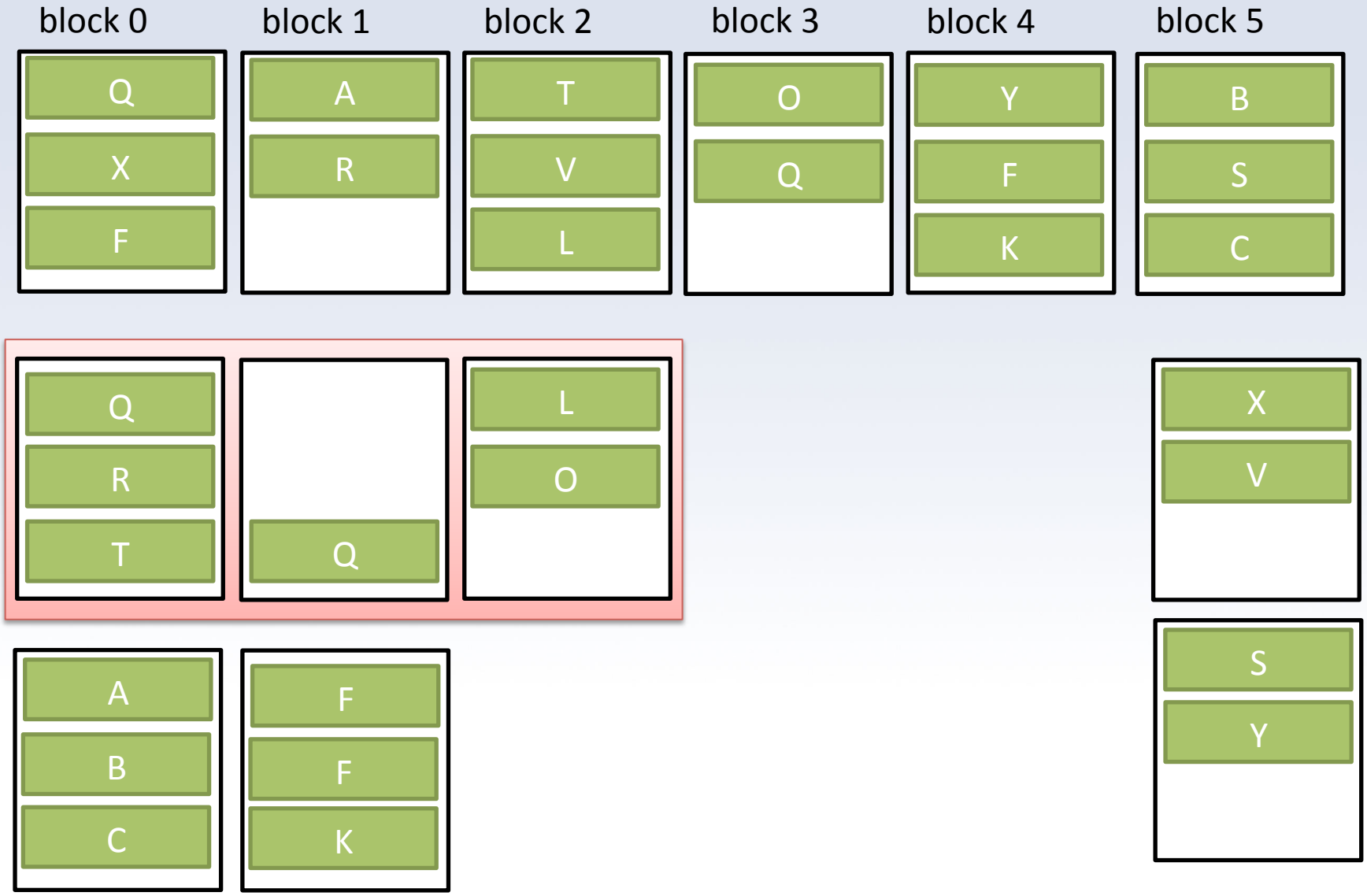
Phase 2


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
L
Q
R
T
X
V
A
B
C
F
F
K
S
Y

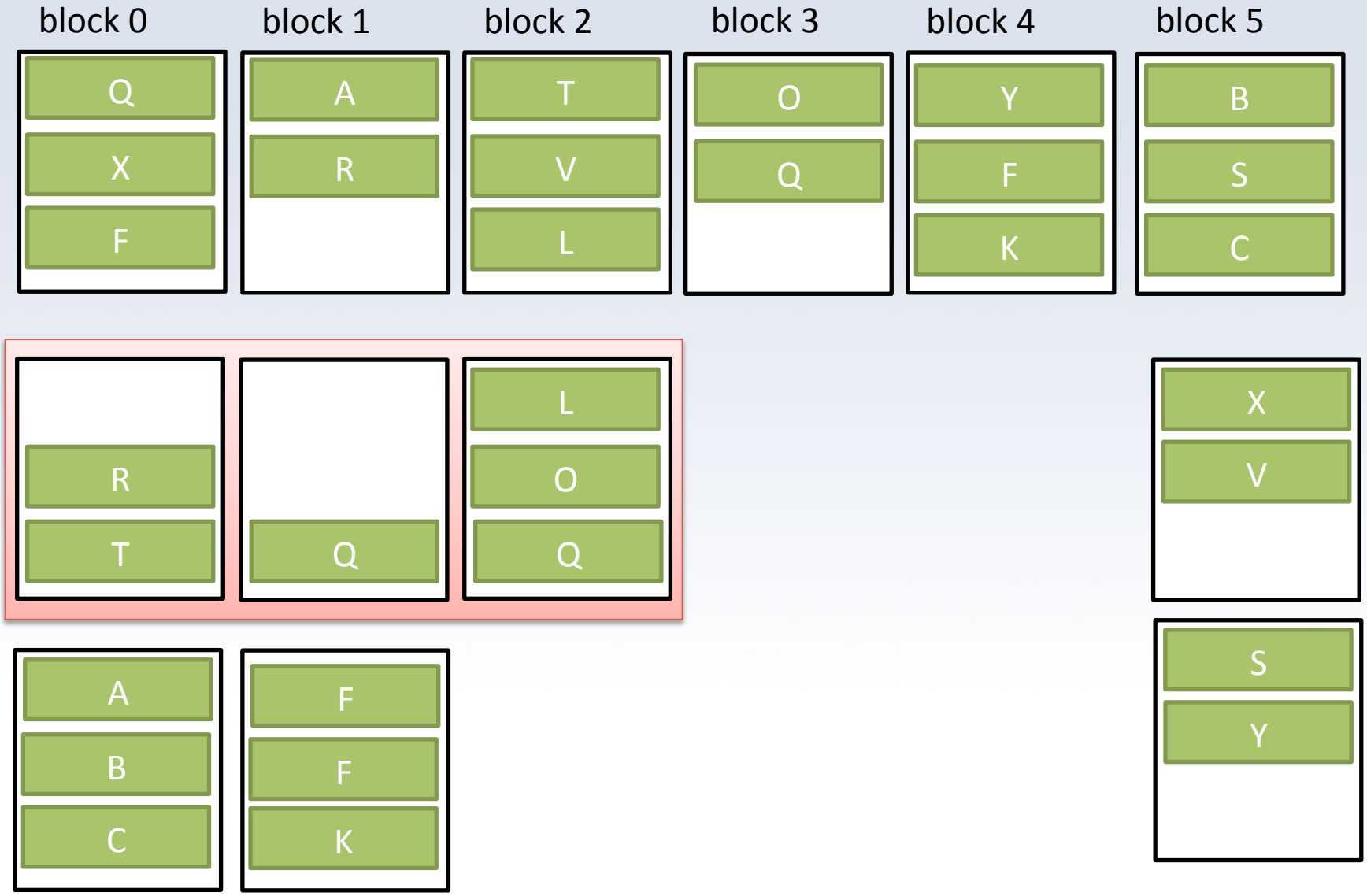
Phase 2


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
L
X
V
S
Y
A
B
C
F
F
K

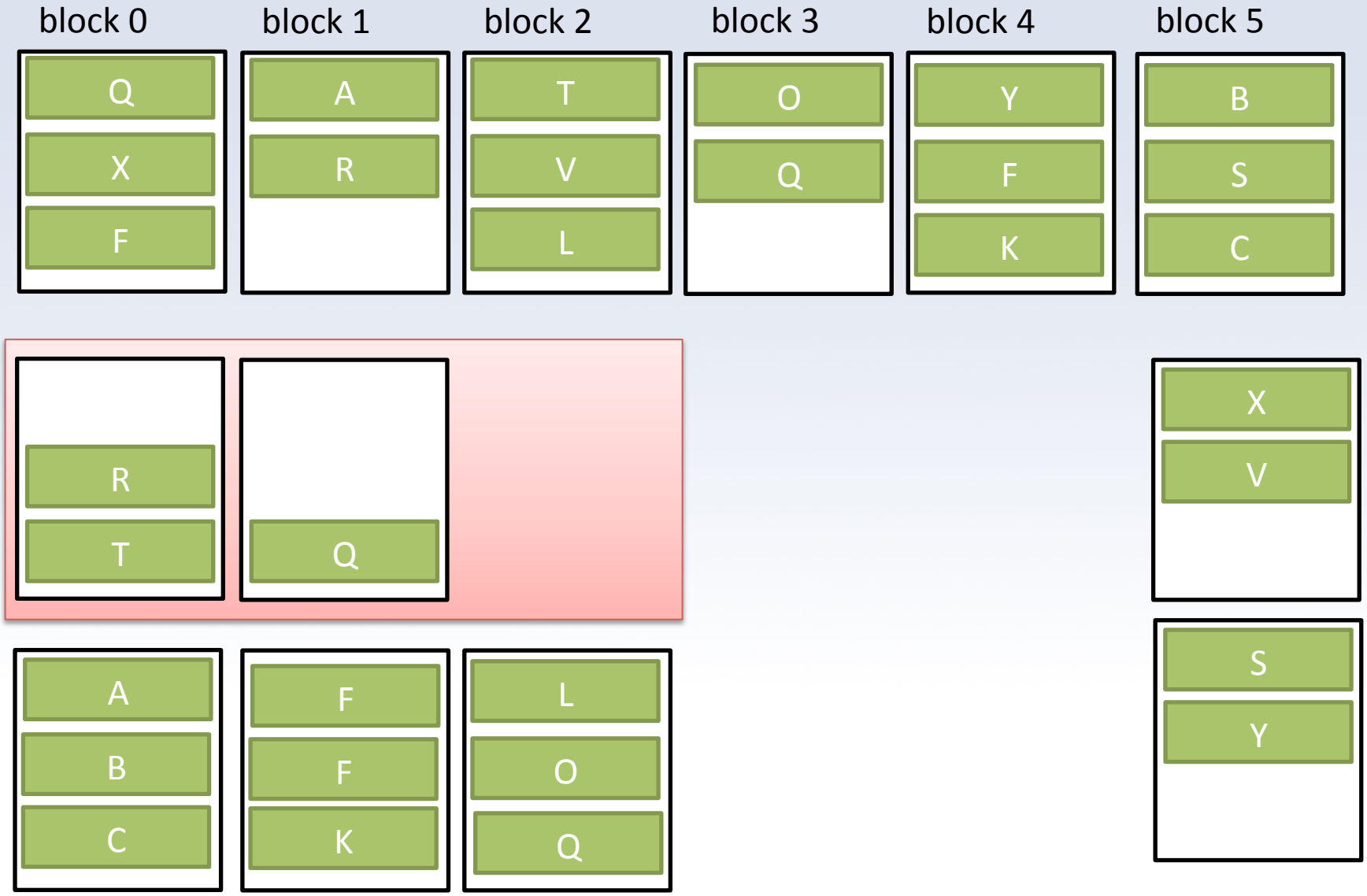
Phase 2


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
L
O
X
V
S
Y
A
B
C
F
F
K

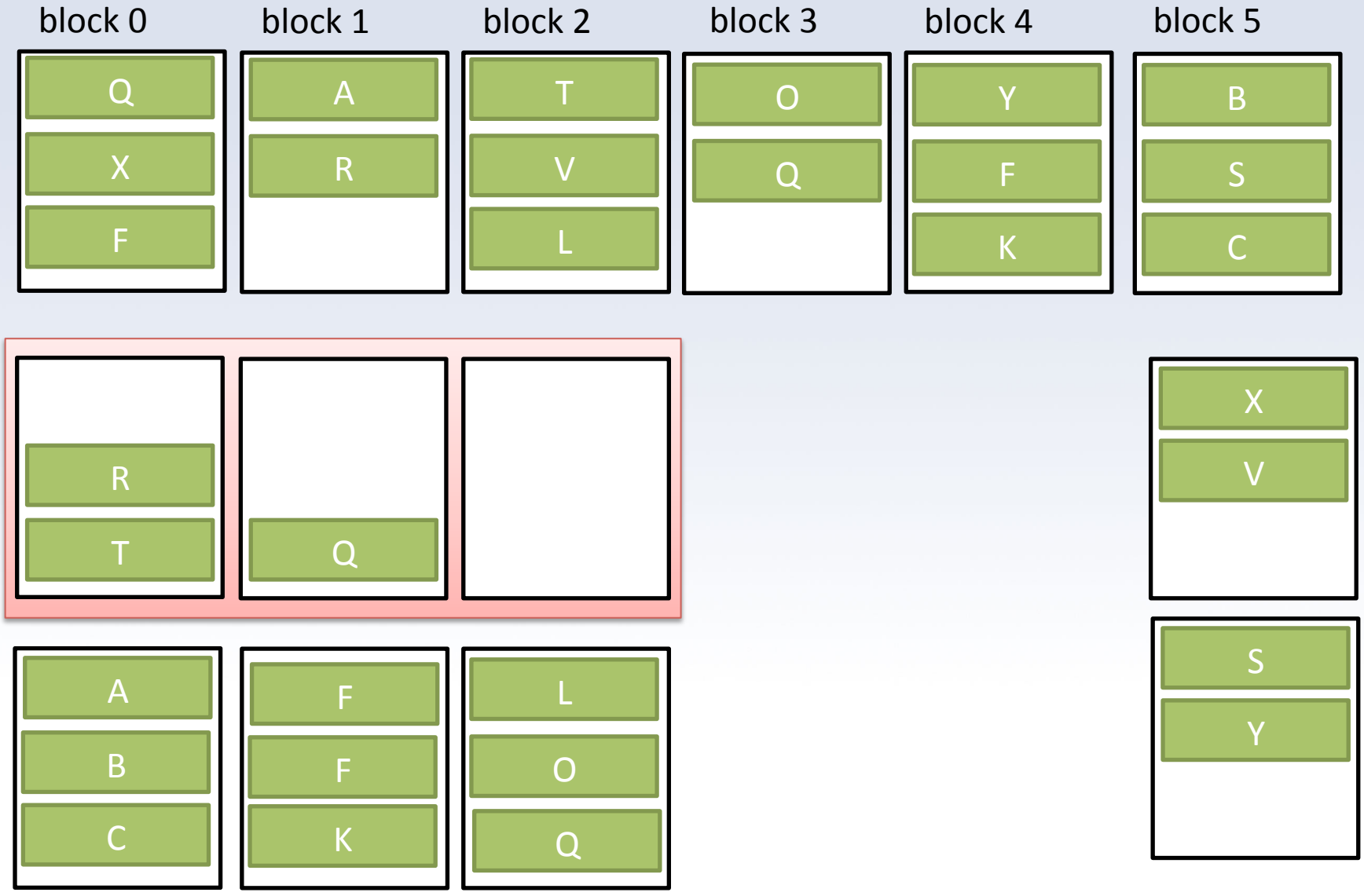
Phase 2


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
L
O
Q
X
V
S
Y
A
B
C
F
F
K

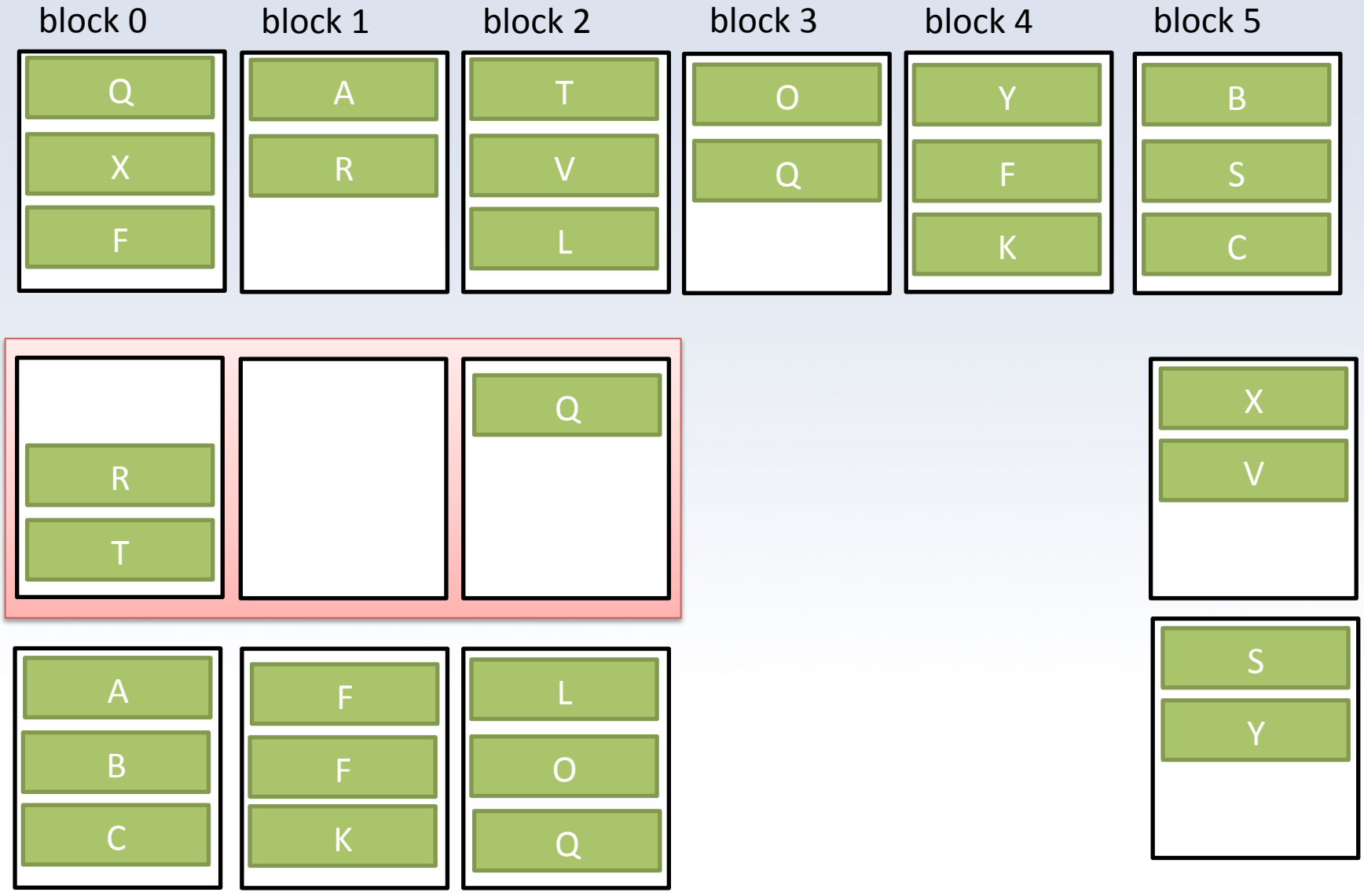
Phase 2


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
X
V
S
Y
A
B
C
F
F
K
L
O
Q

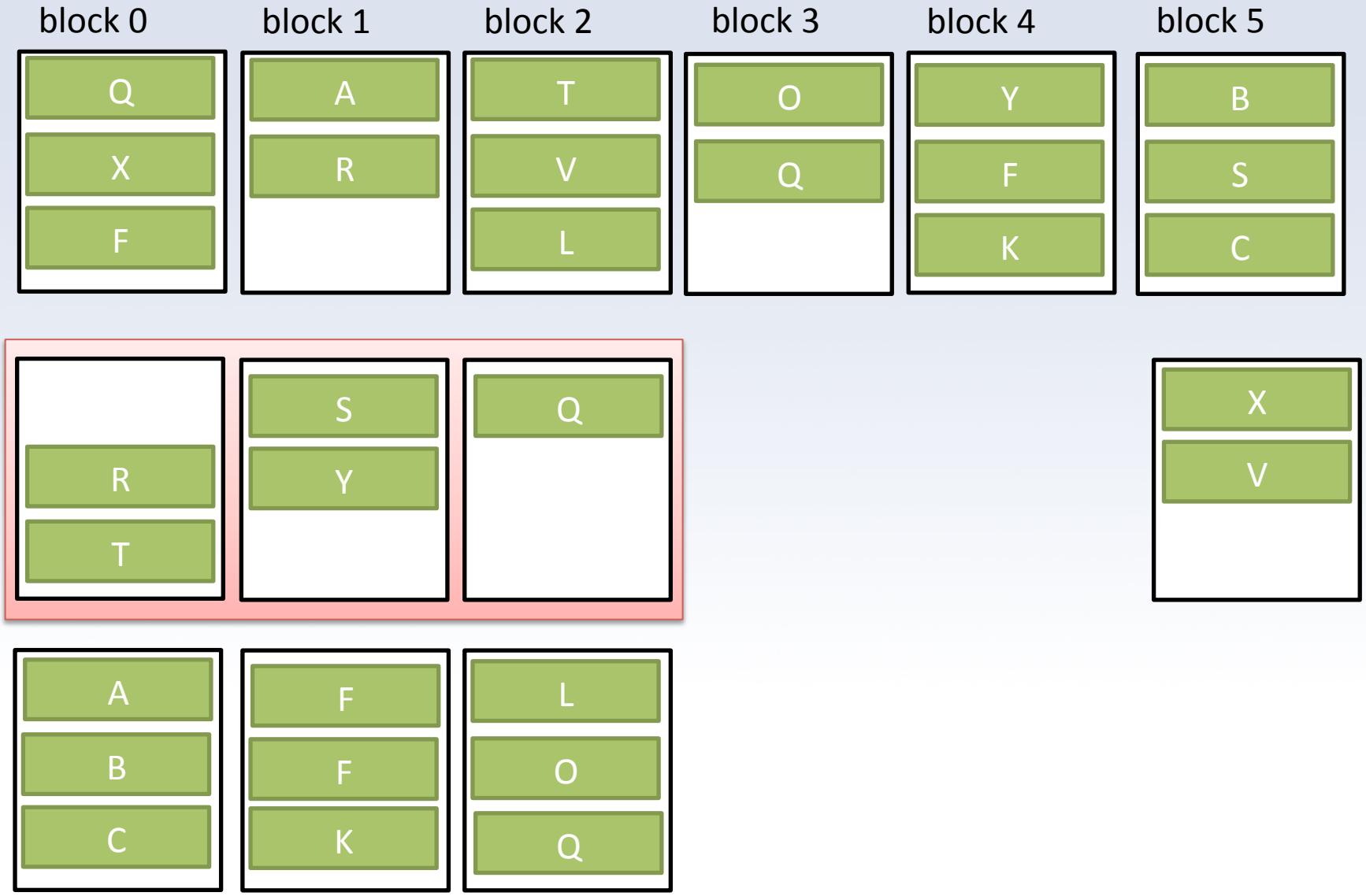
Phase 2


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
X
V
S
Y
A
B
C
F
F
K
L
O
Q

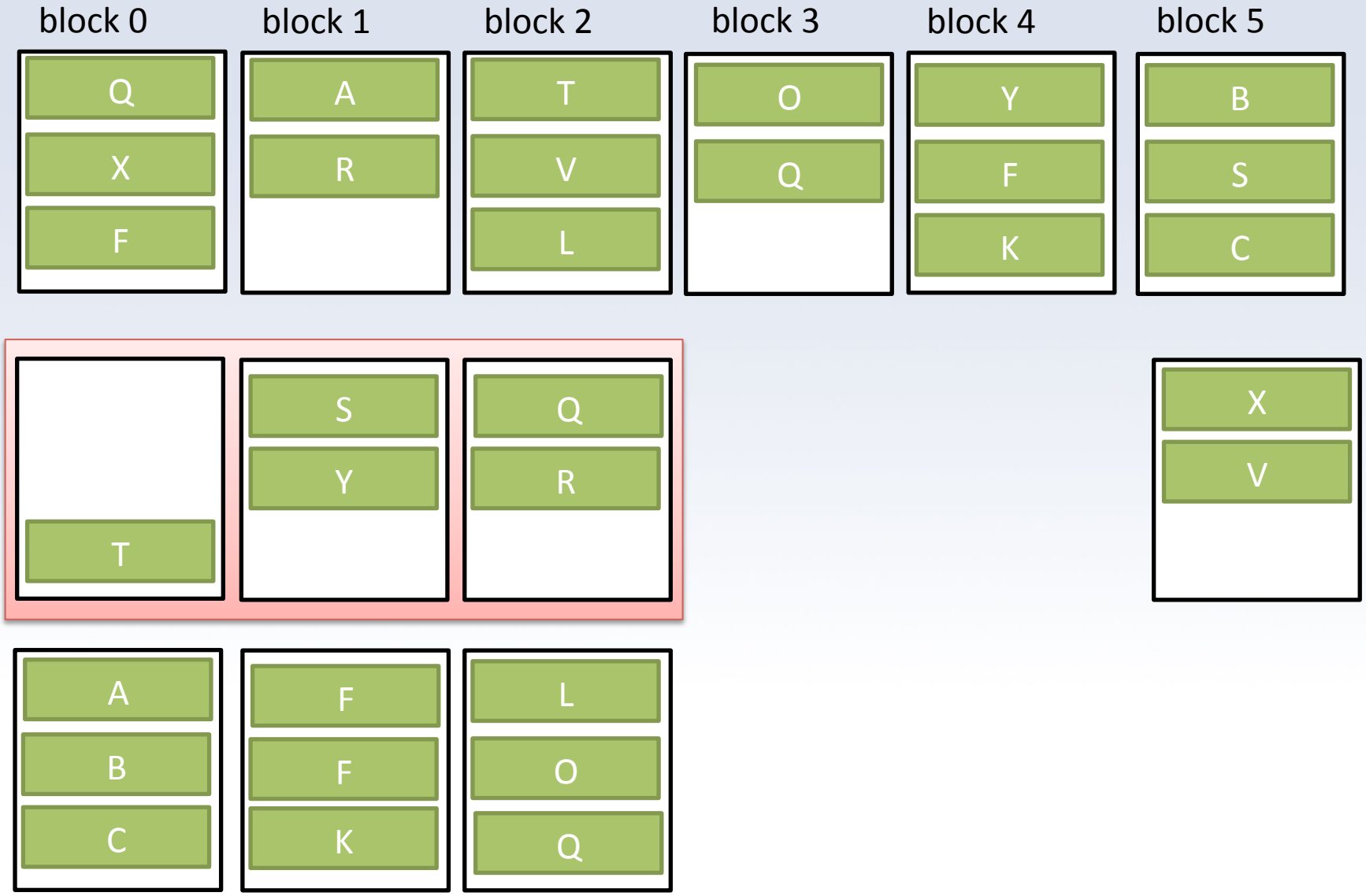
Phase 2


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
X
V
A
B
C
F
F
K
L
O
Q
S
Y

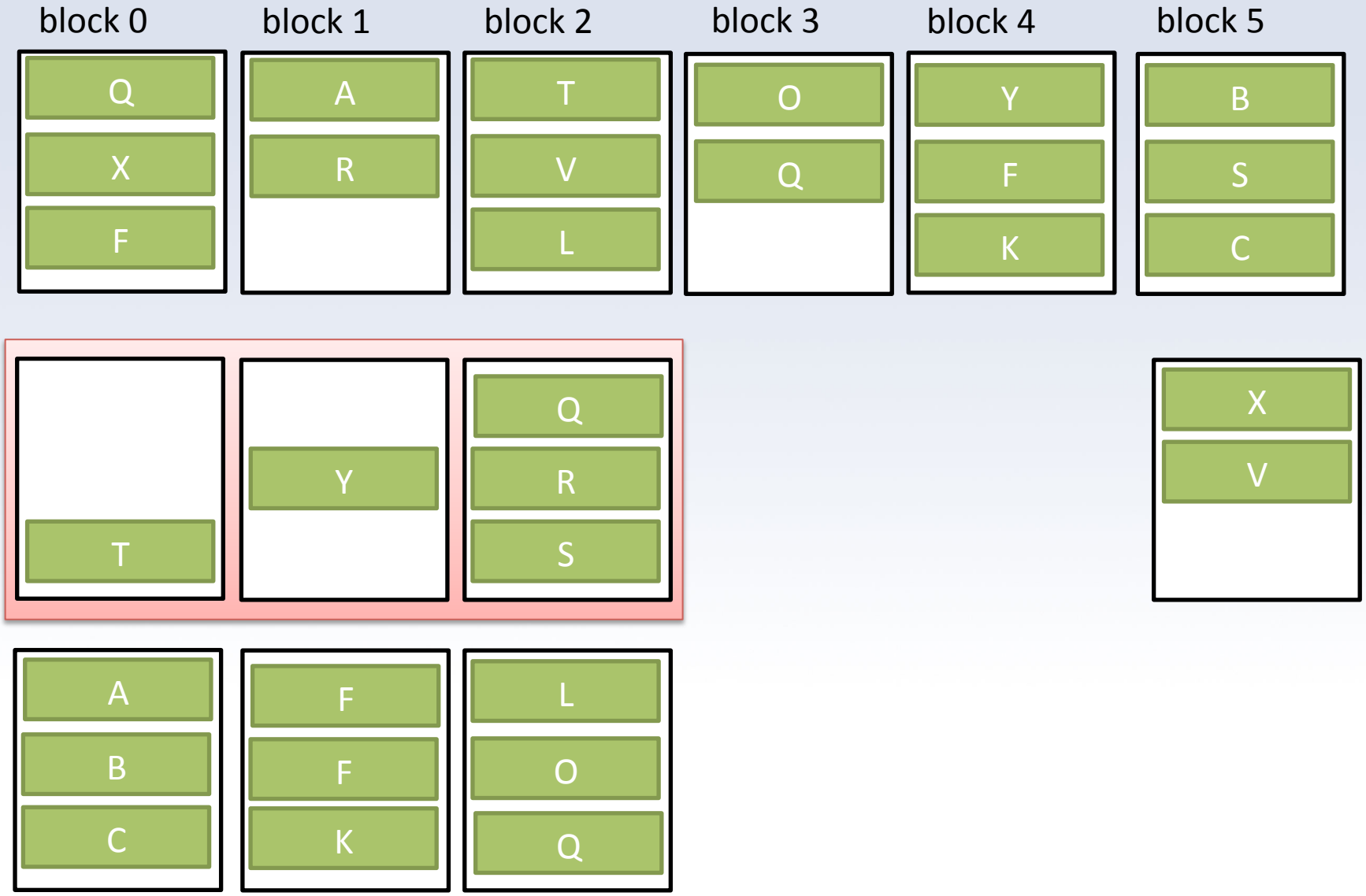
Phase 2


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
R
T
S
Y
Q
X
V
A
B
C
F
F
K
L
O
Q

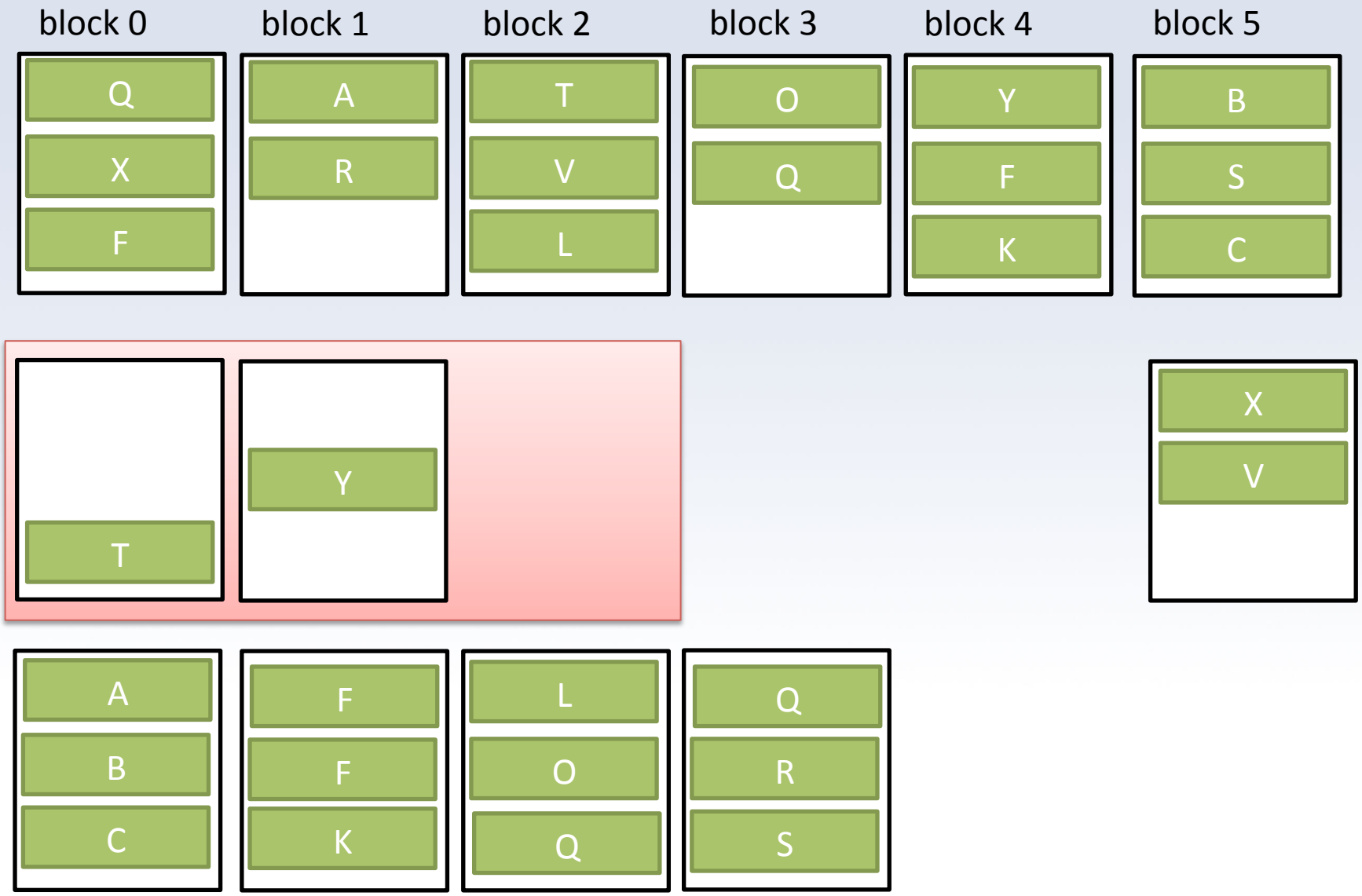
Phase 2


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
Q
R
X
V
A
B
C
F
F
K
L
O
Q

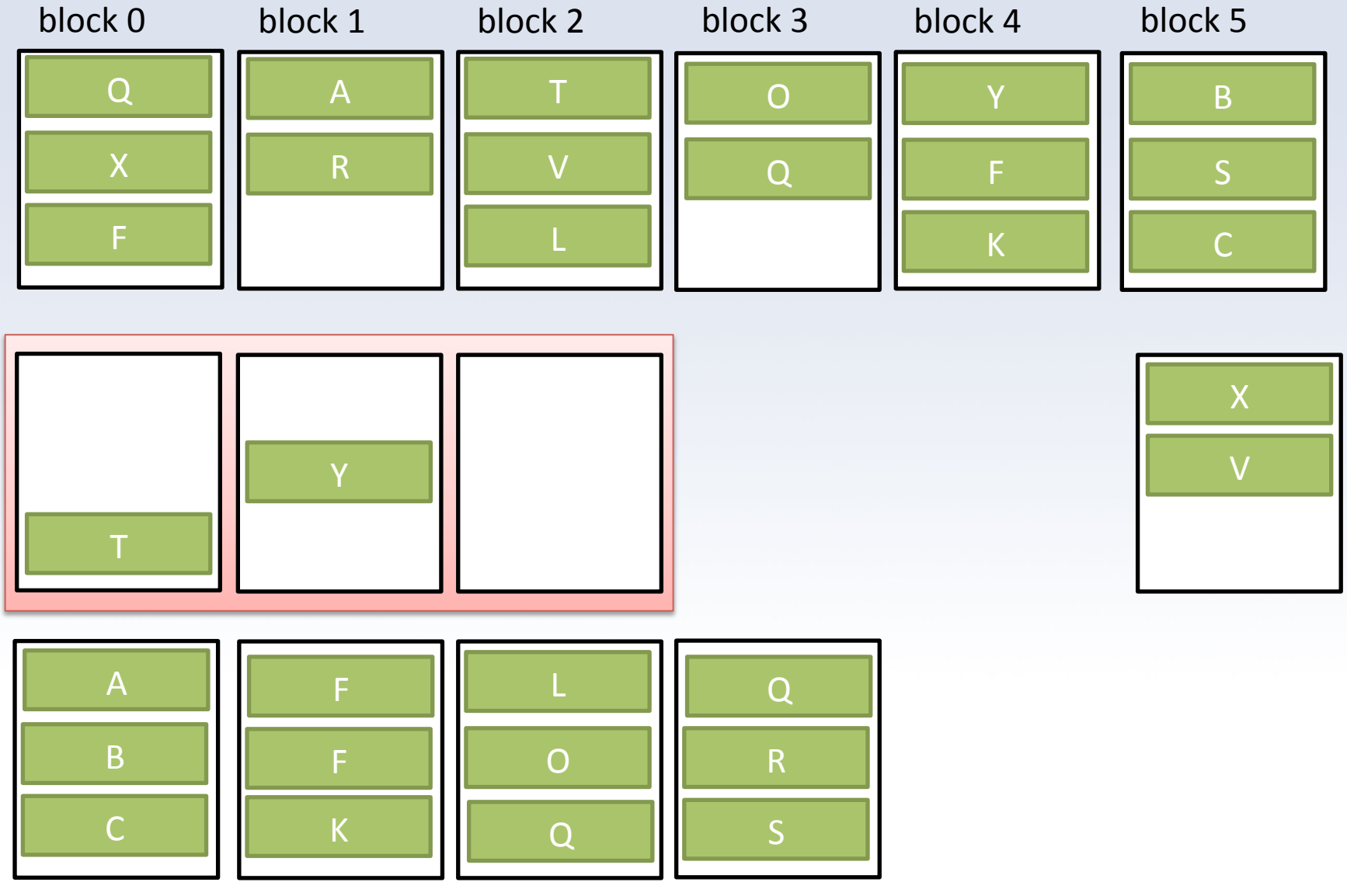
Phase 2


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
Q
R
S
X
V
A
B
C
F
F
K
L
O
Q

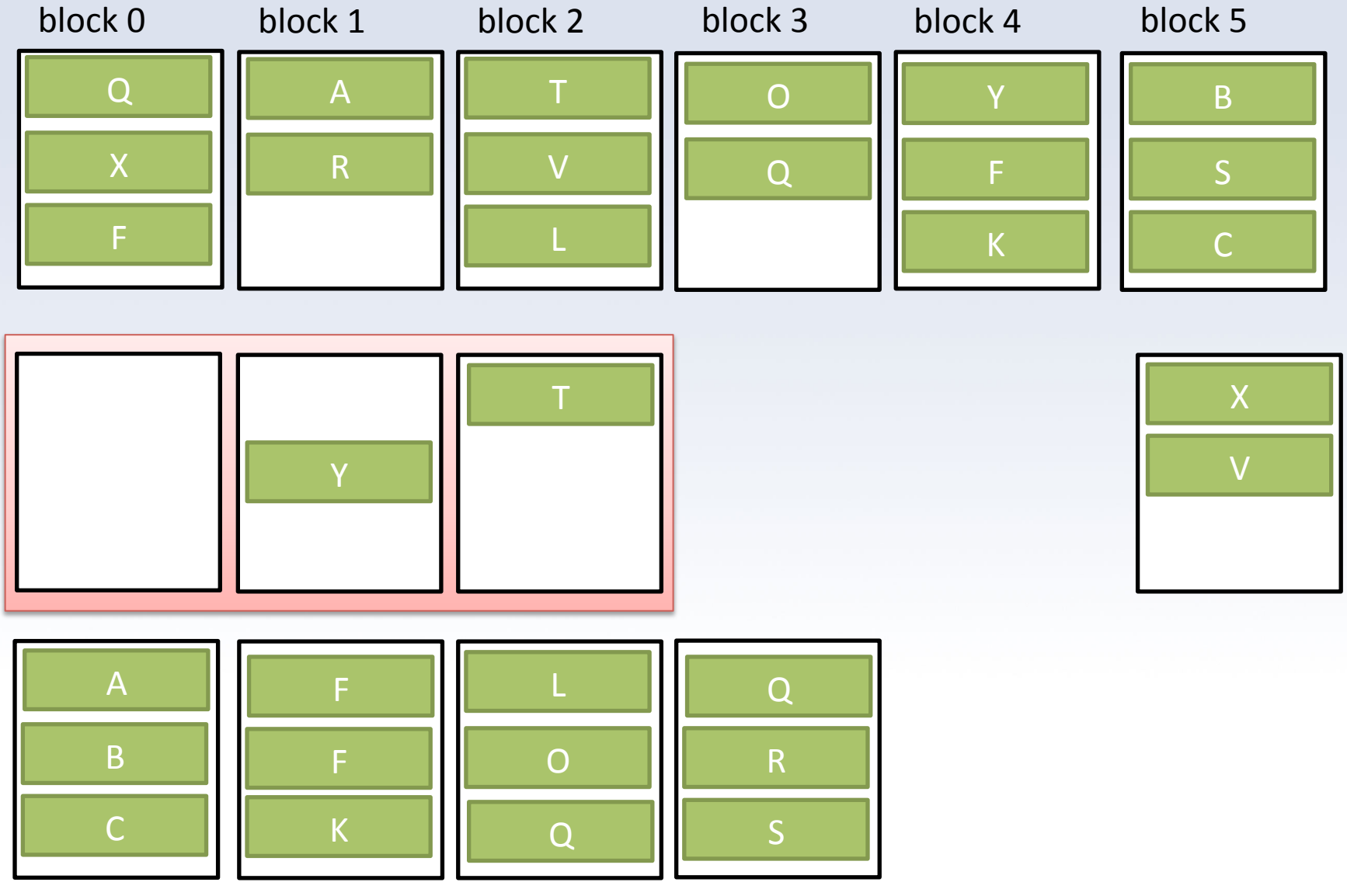
Phase 2


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
X
V
A
B
C
F
F
K
L
O
Q
Q
R
S

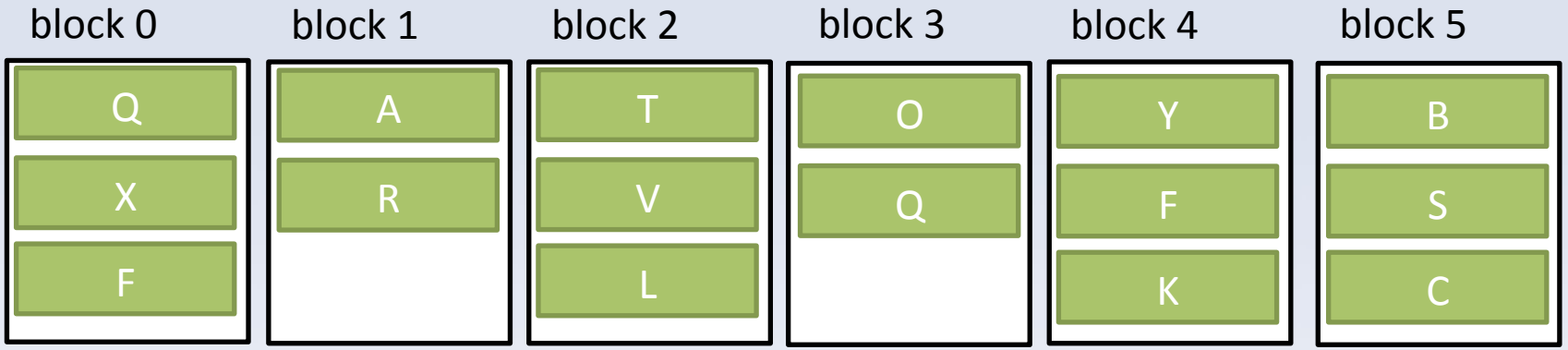
Phase 2

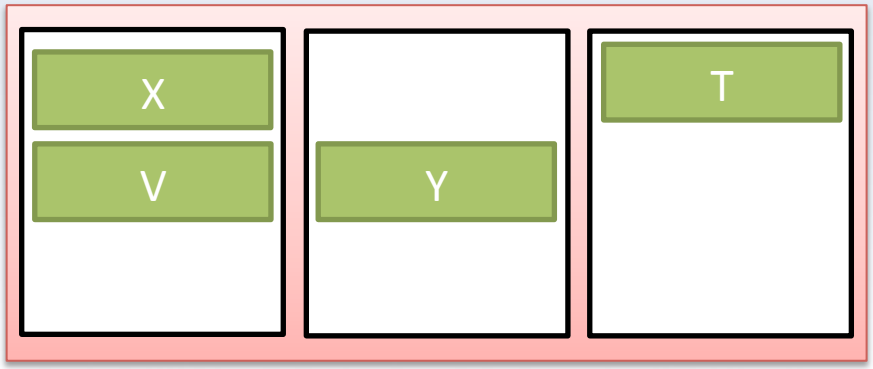

block 0
block 1
block 2
block 3
block 4
block 5
Q
X
F
A
R
T
V
L
O
Q
Y
F
K
B
S
C
T
Y
X
V
A
B
C
F
F
K
L
O
Q
Q
R
S

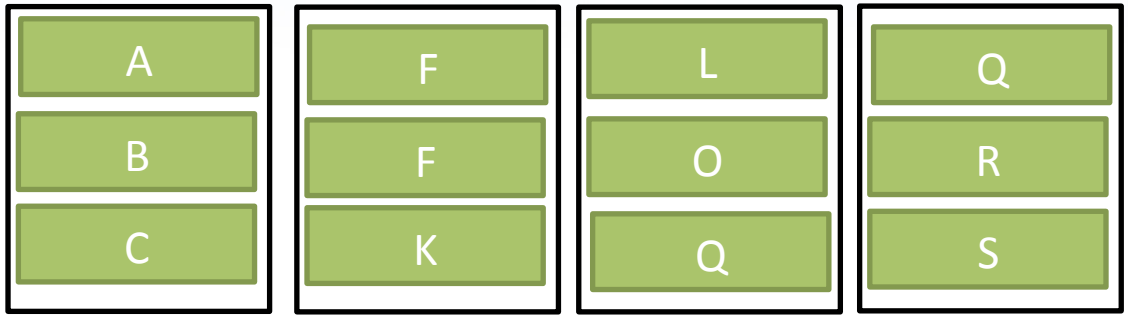
Phase 2


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
T
X
V
A
B
C
F
F
K
L
O
Q
Q
R
S

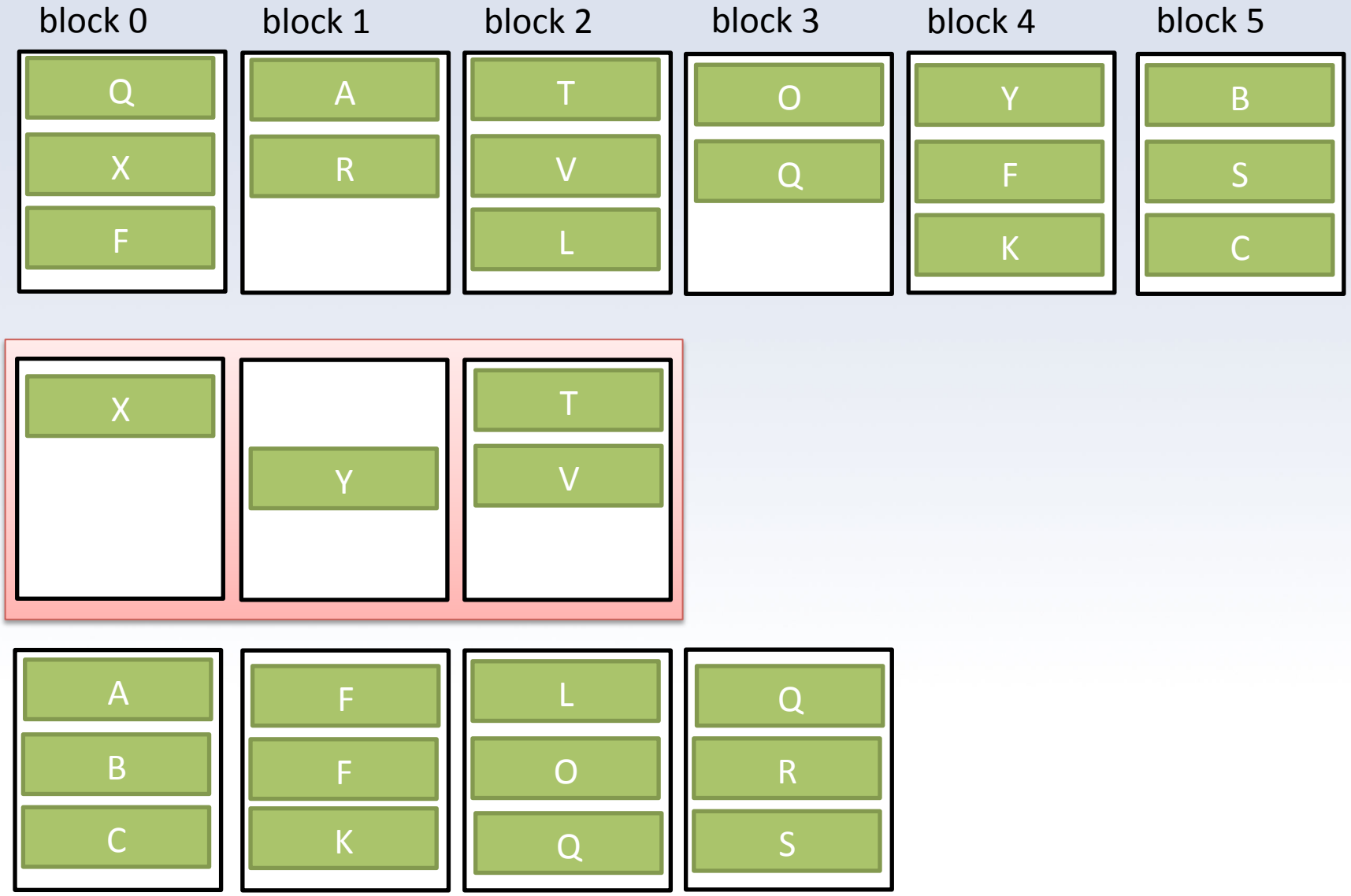
Phase 2


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

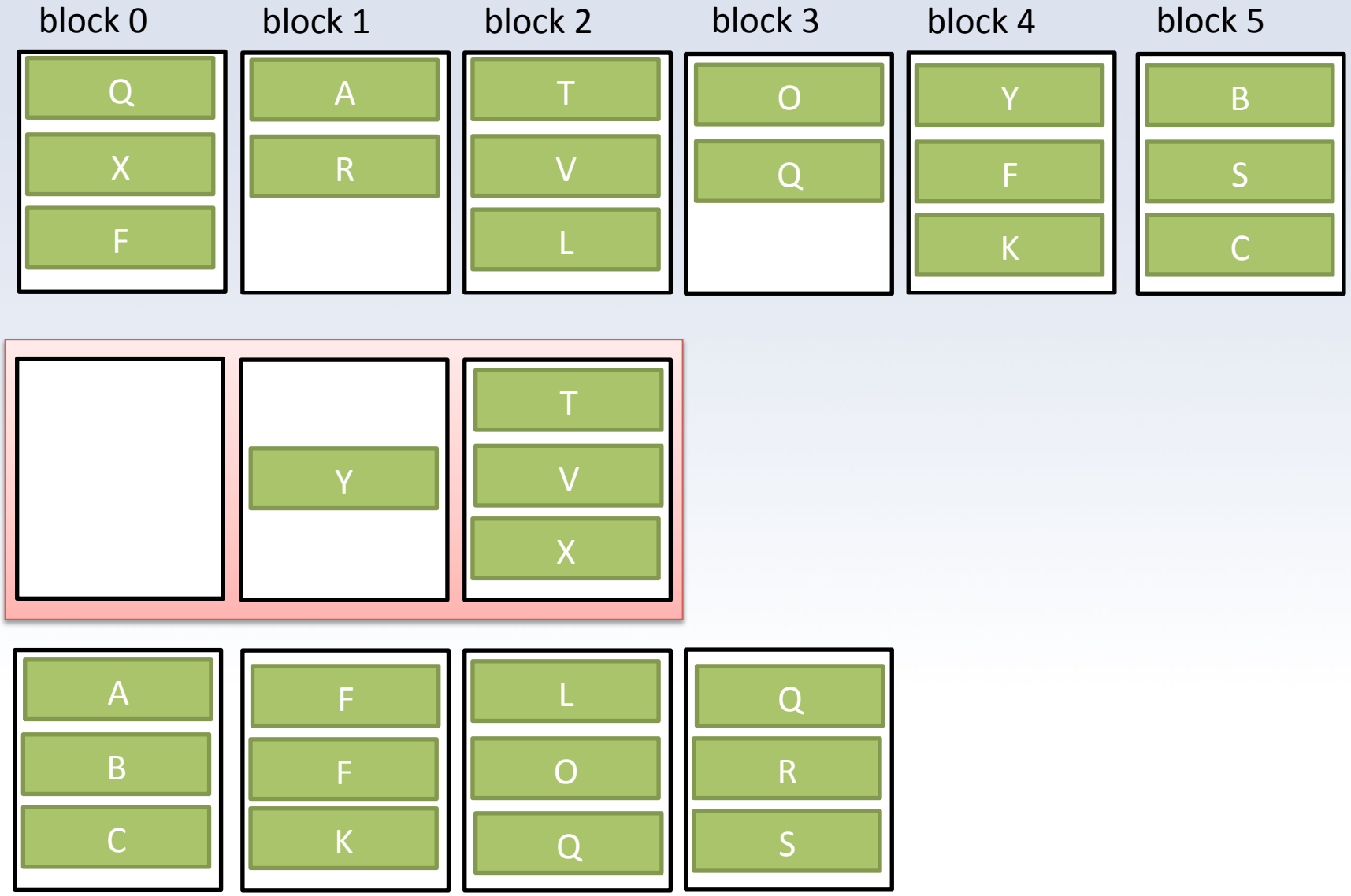

X
V
Y
T


A
B
C
F
F
K
L
O
Q
Q
R
S

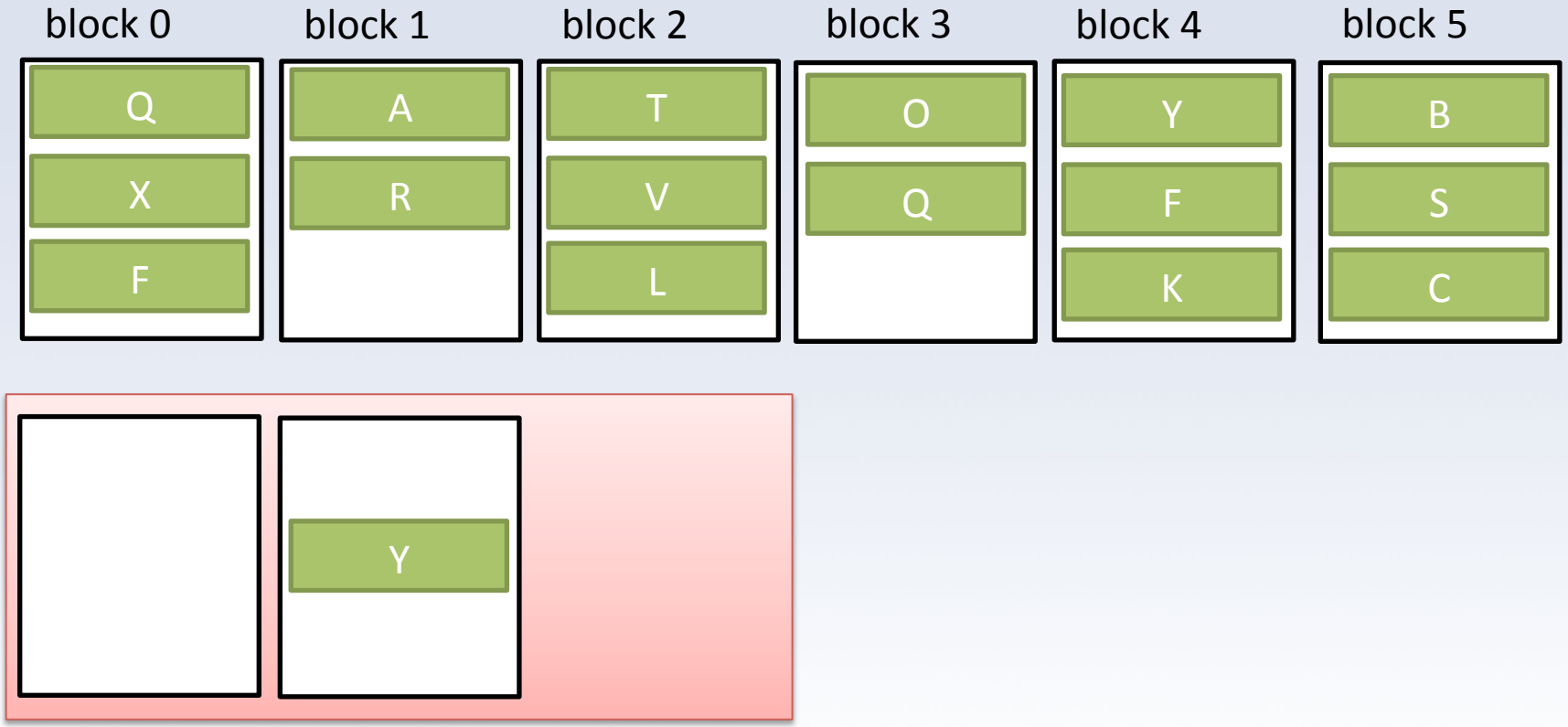
Phase 2


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
T
V
A
B
C
F
F
K
L
O
Q
Q
R
S

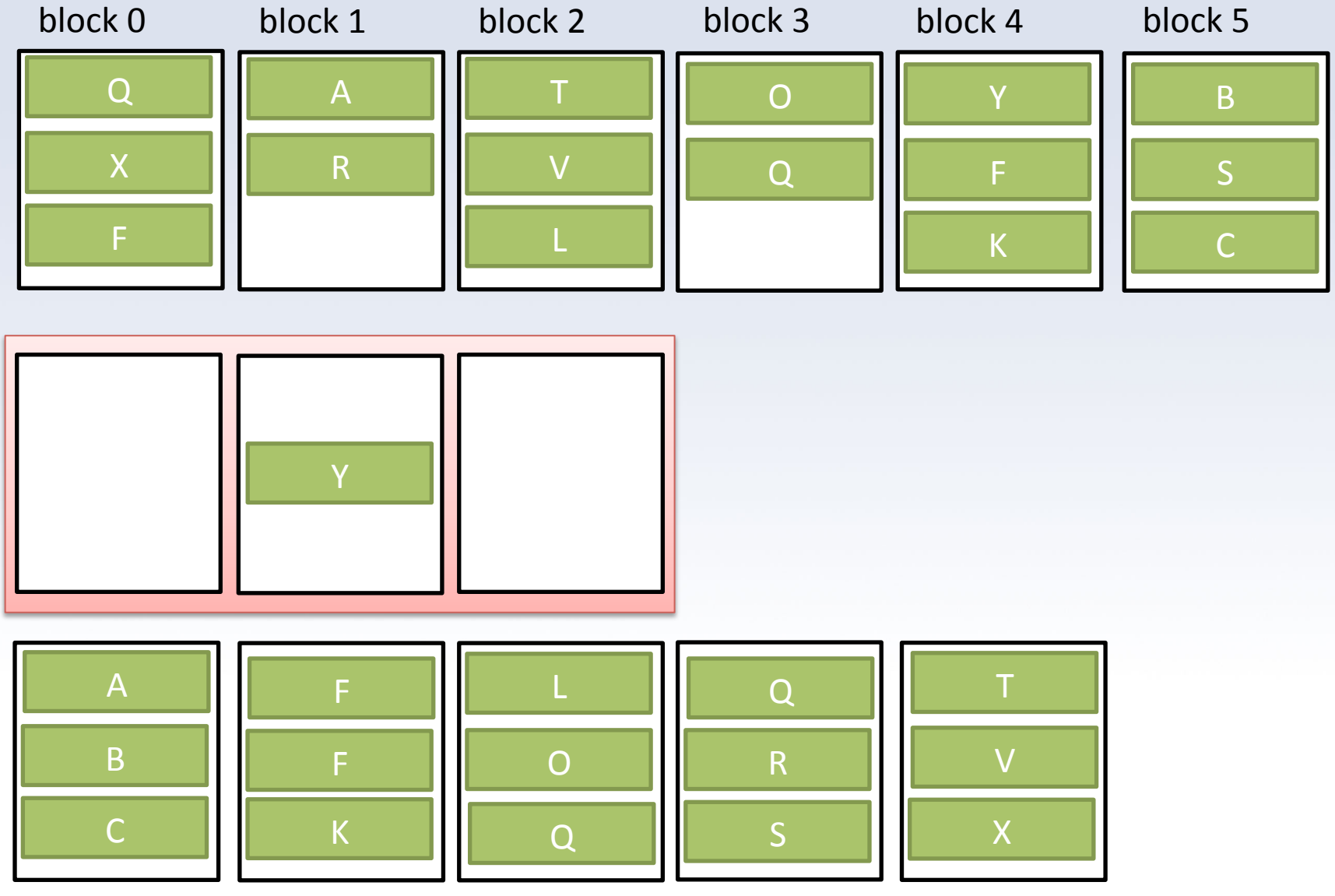
Phase 2


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
T
V
X
A
B
C
F
F
K
L
O
Q
Q
R
S

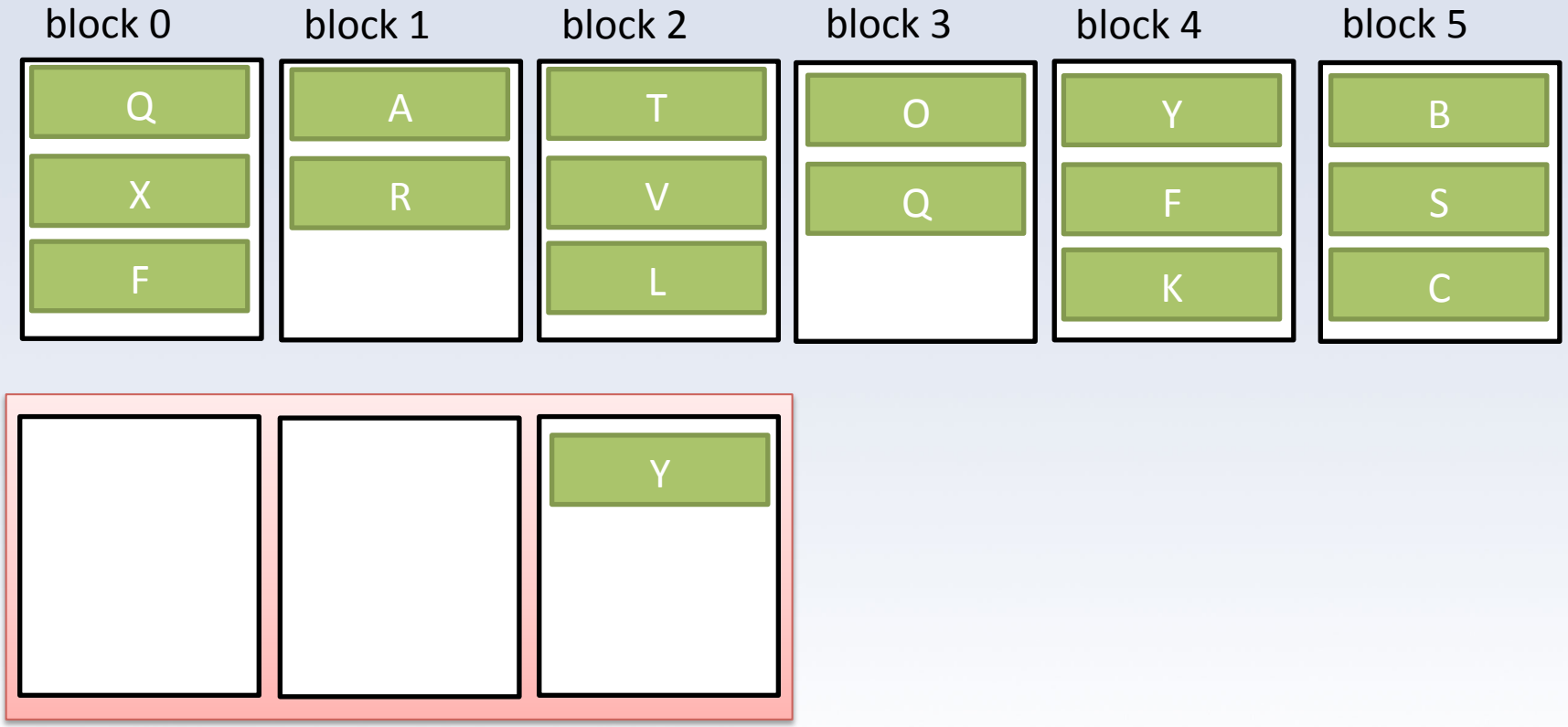
Phase 2


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
A
B
C
F
F
K
L
O
Q
Q
R
S
T
V
X

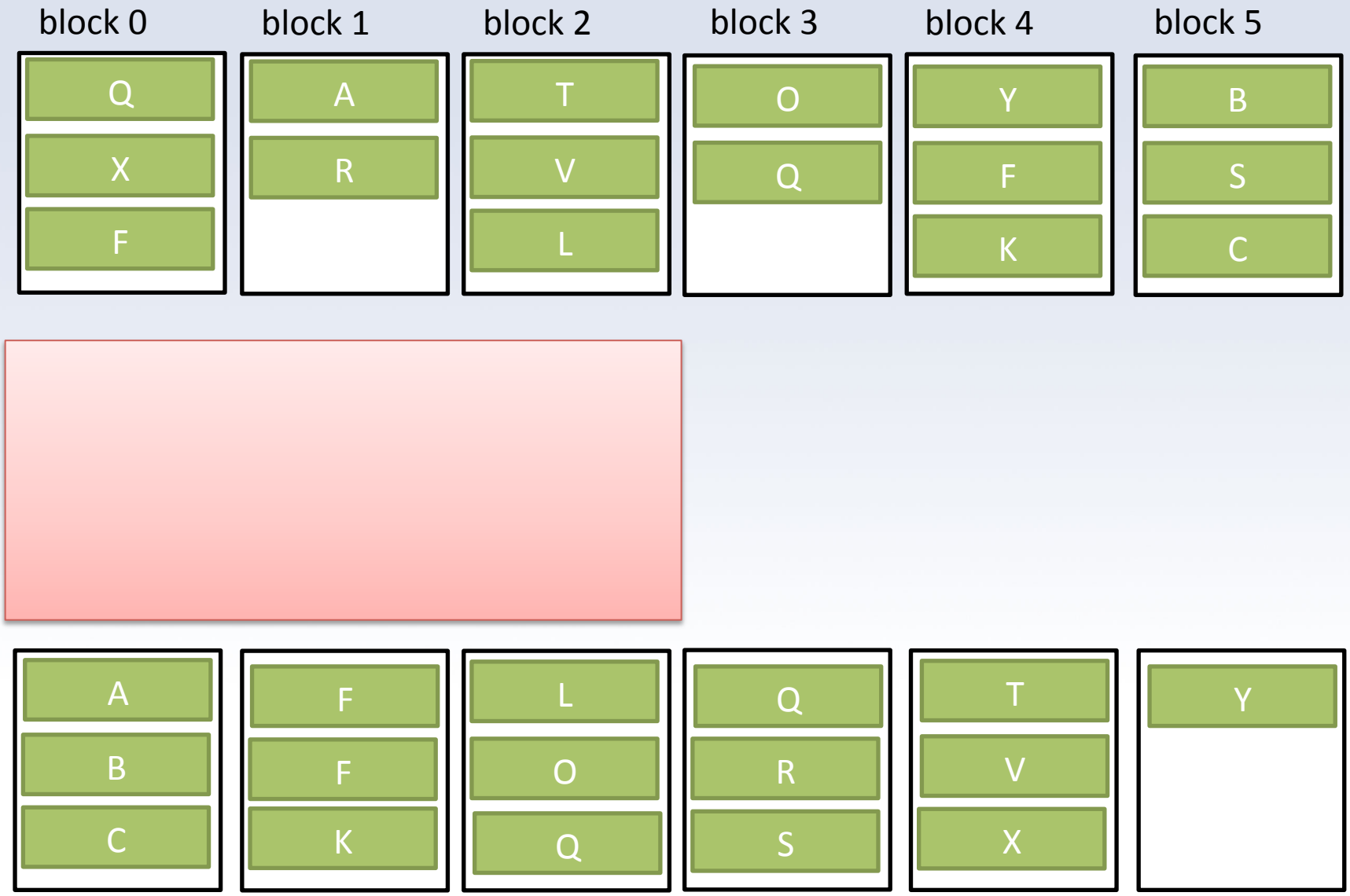
Phase 2


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
A
B
C
F
F
K
L
O
Q
Q
R
S
T
V
X

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
A
B
C
F
F
K
L
O
Q
Q
R
S
T
V
X

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
A
B
C
F
F
K
L
O
Q
Q
R
S
T
V
X
Y

Analysis

- Number of sublists is B/M
- To fit one block for each sublist, $B/M \leq M-1$
- Rewritten, $B \leq M(M-1)$
- Requires that $\sqrt{B(R)} \leq M$

Analysis

- Read B(R) blocks in Phase 1
- Write B(R) blocks in Phase 1
- Read B(R) blocks in Phase 2
- Write B(R) blocks in Phase 2

Analysis

- Read B(R) blocks in Phase 1
- Write B(R) blocks in Phase 1
- Read B(R) blocks in Phase 2
- ~~Write B(R) blocks in Phase 2~~ Output

Analysis

- Read B(R) blocks in Phase 1
- Write B(R) blocks in Phase 1
- Read B(R) blocks in Phase 2
- ~~Write B(R) blocks in Phase 2~~ Output
- Cost is therefore 3B(R)

Sorting-based δ

- We can implement δ operation with our TPMMS as follows:
  1. Create all of the sorted sublists as usual
  2. In phase 2, don't output duplicate entries (keep track of last record output and skip identical records)

Sorting-based γ

- We can implement γ operation with our TPMMS as follows:
  1. Create all of the sorted sublists as usual
  2. In phase 2, don't output duplicate entries. Instead, compute aggregation functions on duplicate entries

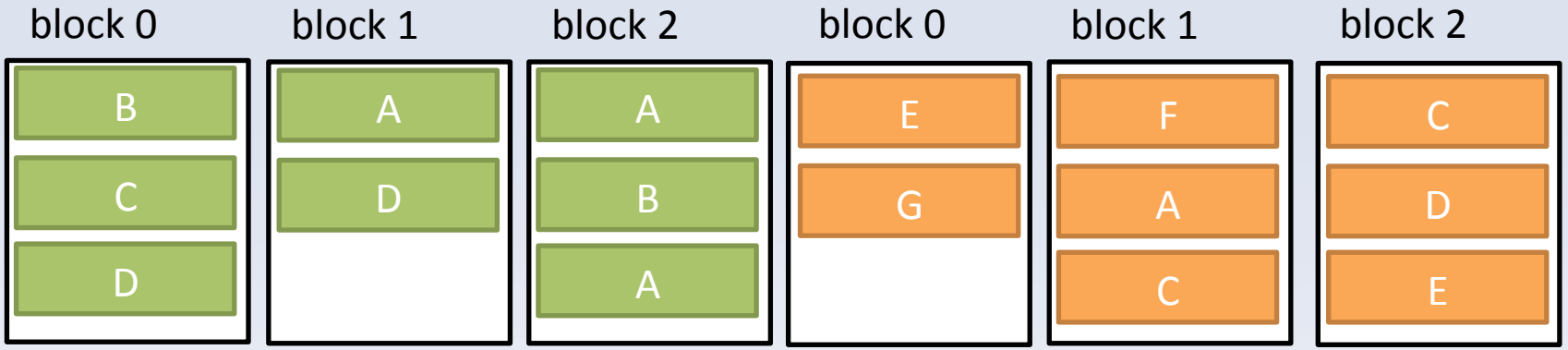
Sorting-based ∪

- We can implement ∪ operation with our TPMMS as follows:
  1. Create all of the sorted sublists for both R *and* S
  2. In phase 2, bring sorted lists for BOTH relations into buffers. Don't output duplicate entries.
- Essentially, treat R and S as one relation

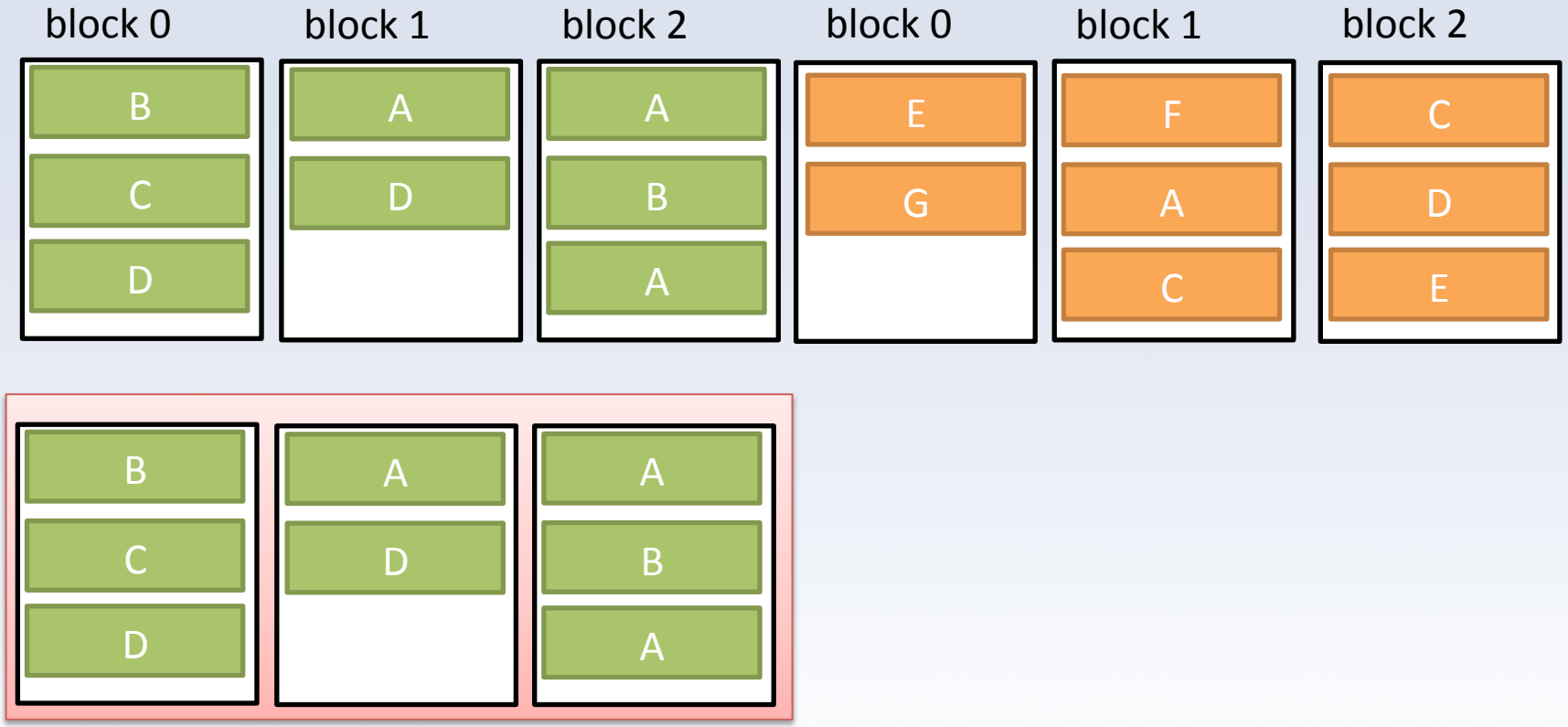
Sorting-based ∩

- We can implement ∩ operation with our TPMMS as follows:
  1. Create all of the sorted sublists for both R *and* S
  2. In phase 2, bring sorted lists for both relations into buffers, but keep them separate
  3. Only output when both relations contain the same value

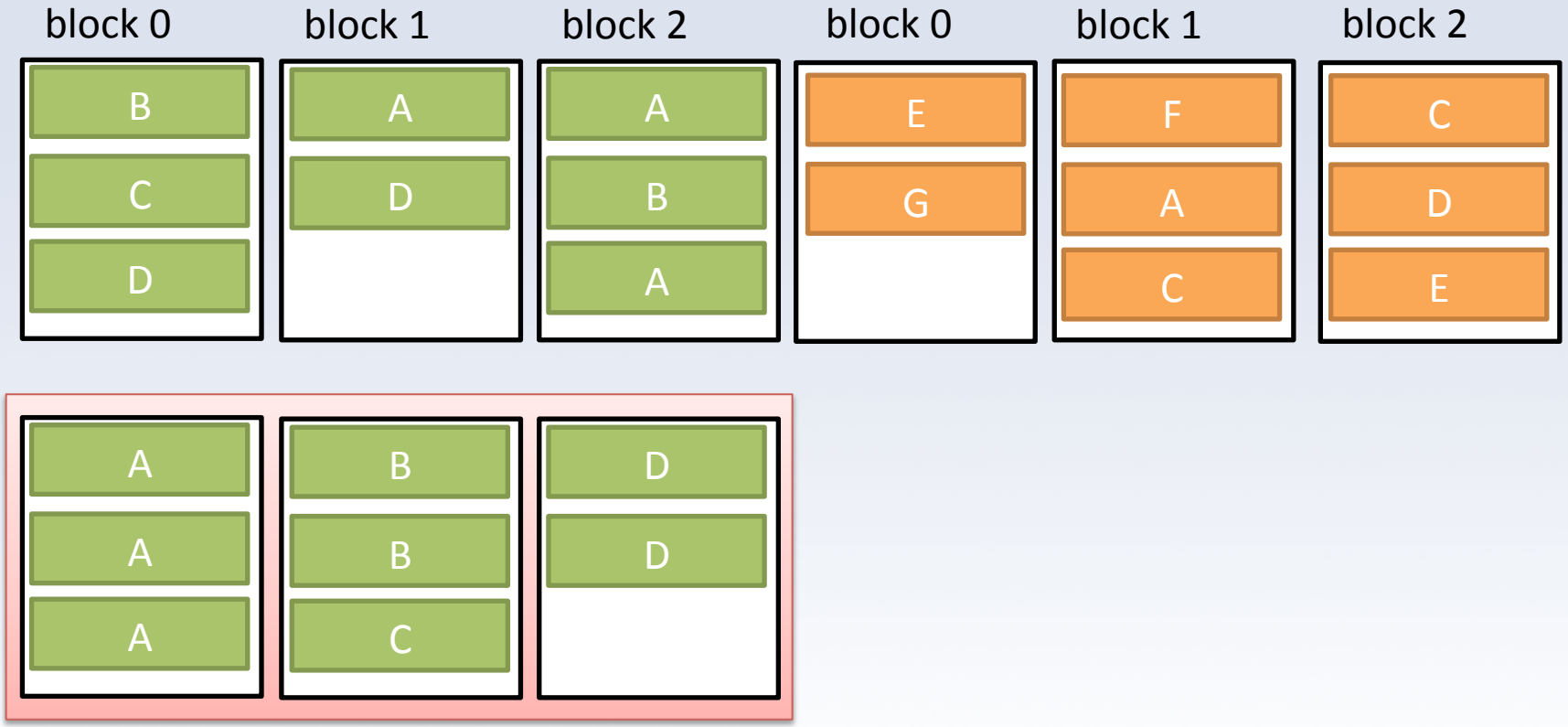
Example R ∩ S


block 0
B
C
D
block 1
A
D
block 2
A
B
A
block 0
E
G
block 1
F
A
C
block 2
C
D
E

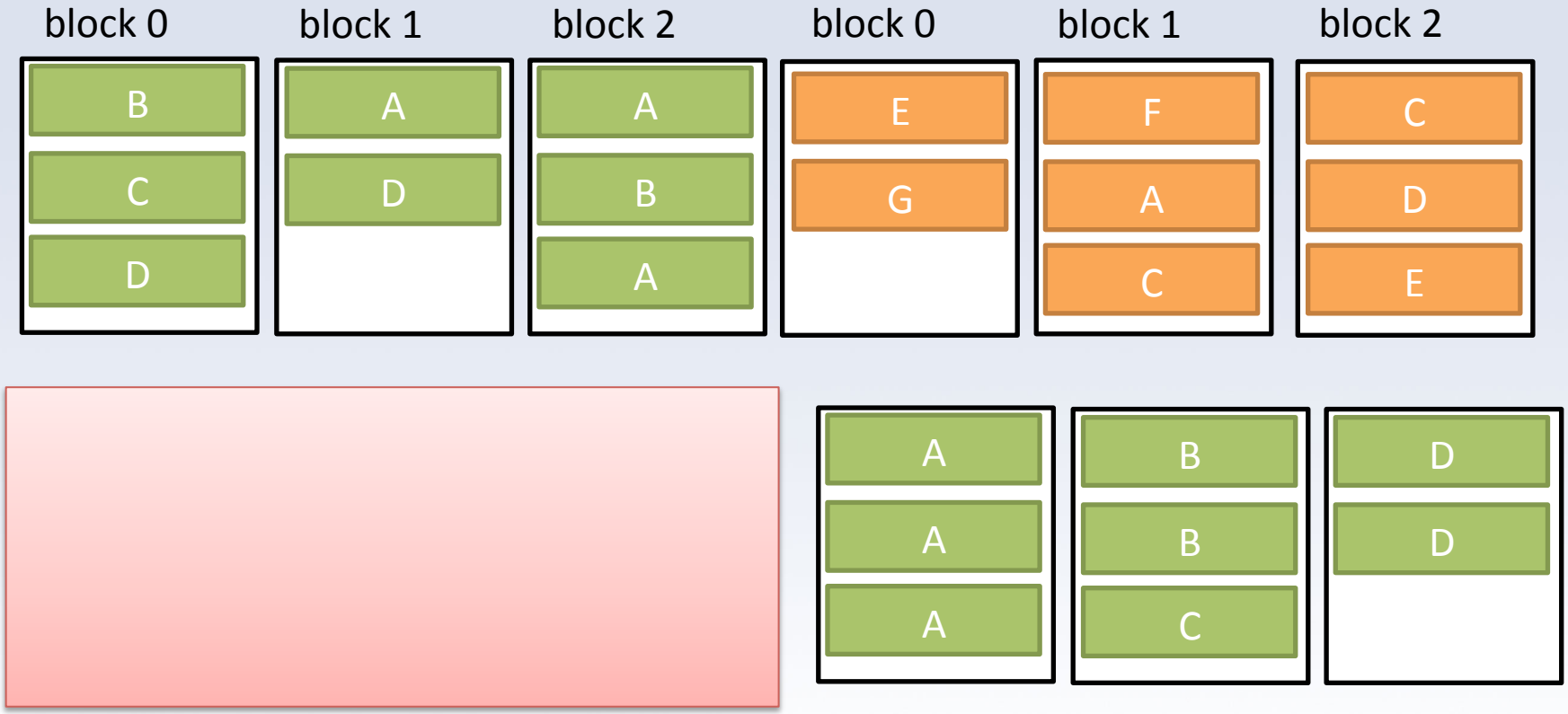
Example R ∩ S


block 0
B
C
D
block 1
A
D
block 2
A
B
A
block 0
E
G
block 1
F
A
C
block 2
C
D
E
B
C
D
A
D
A
B
A

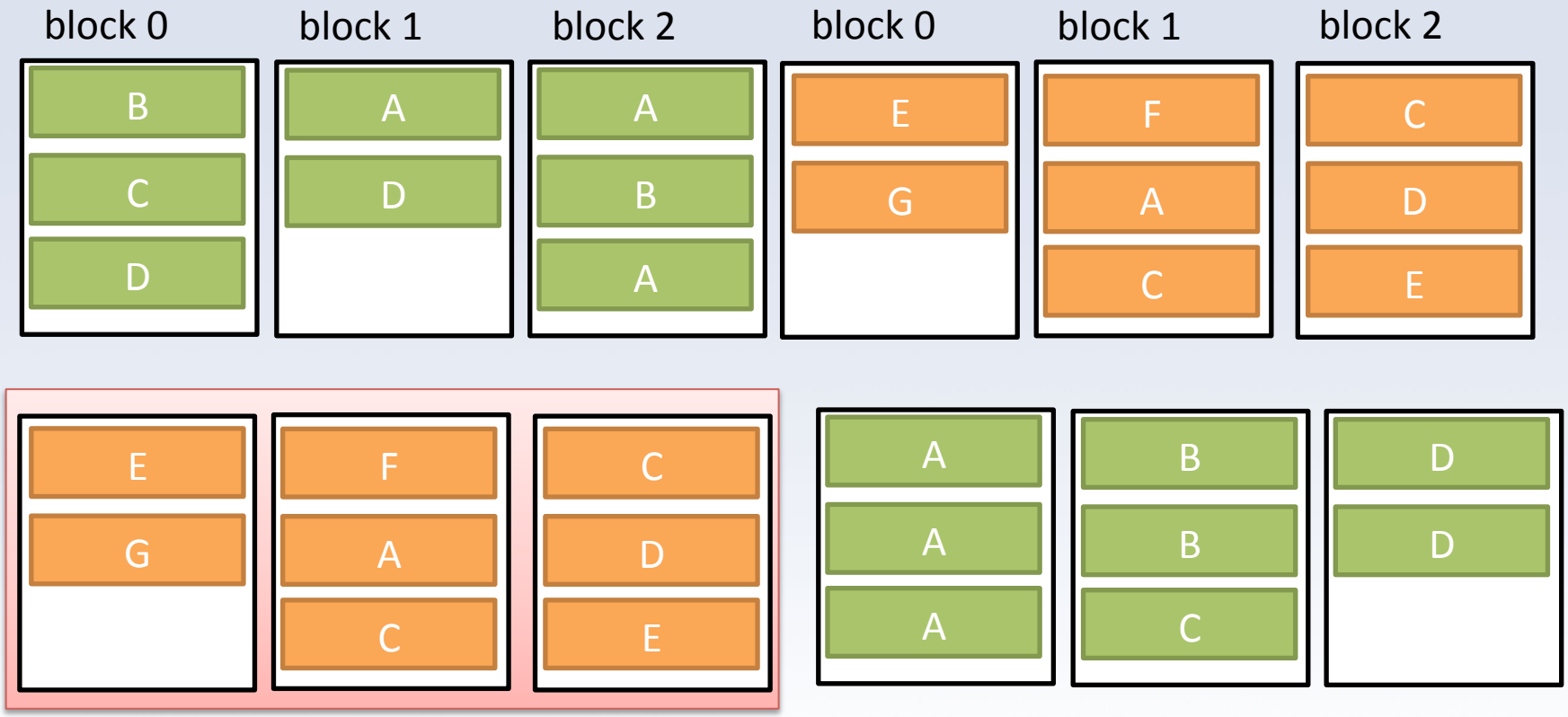
Example R ∩ S


block 0
B
C
D
block 1
A
D
block 2
A
B
A
block 0
E
G
block 1
F
A
C
block 2
C
D
E
A
A
A
B
B
C
D
D

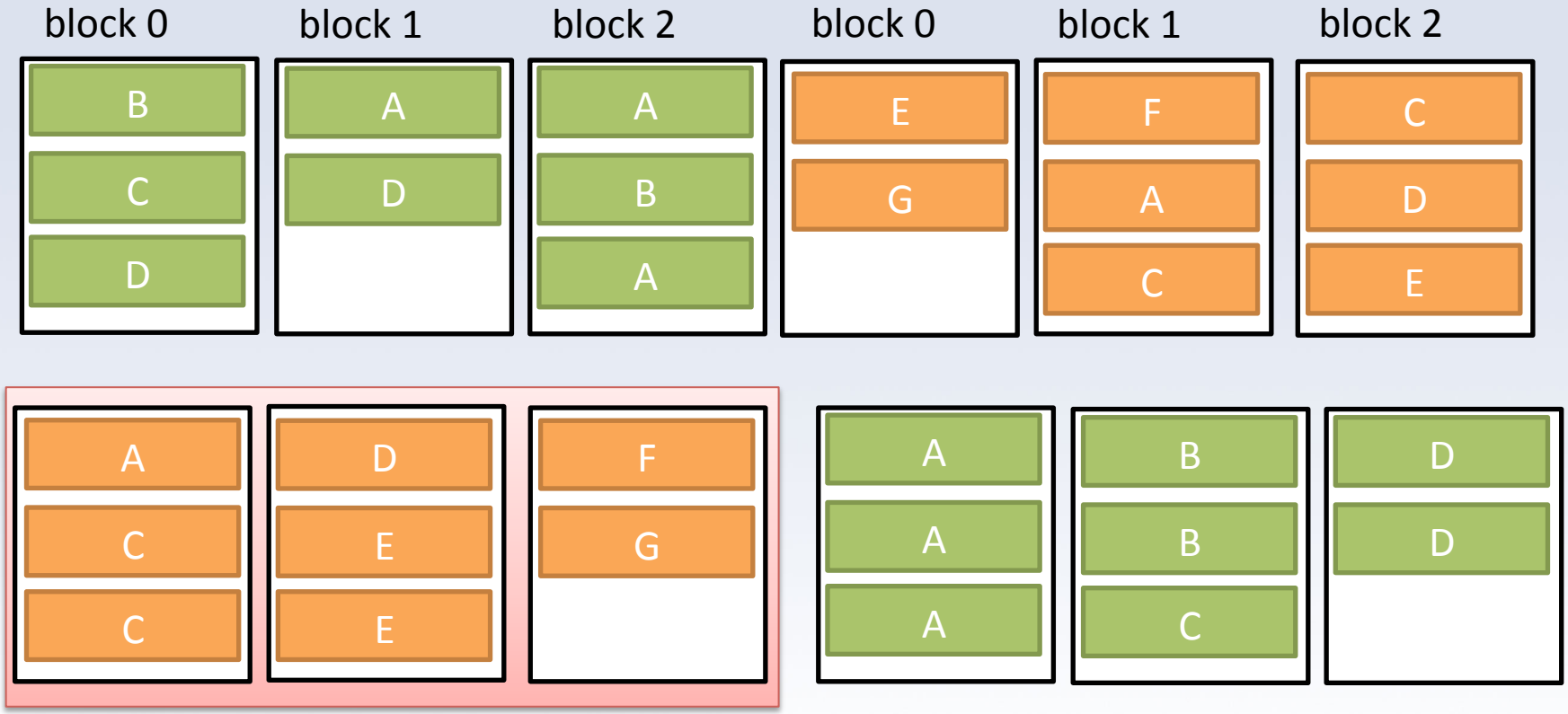
Example R ∩ S


block 0
B
C
D
block 1
A
D
block 2
A
B
A
block 0
E
G
block 1
F
A
C
block 2
C
D
E
A
A
A
B
B
C
D
D

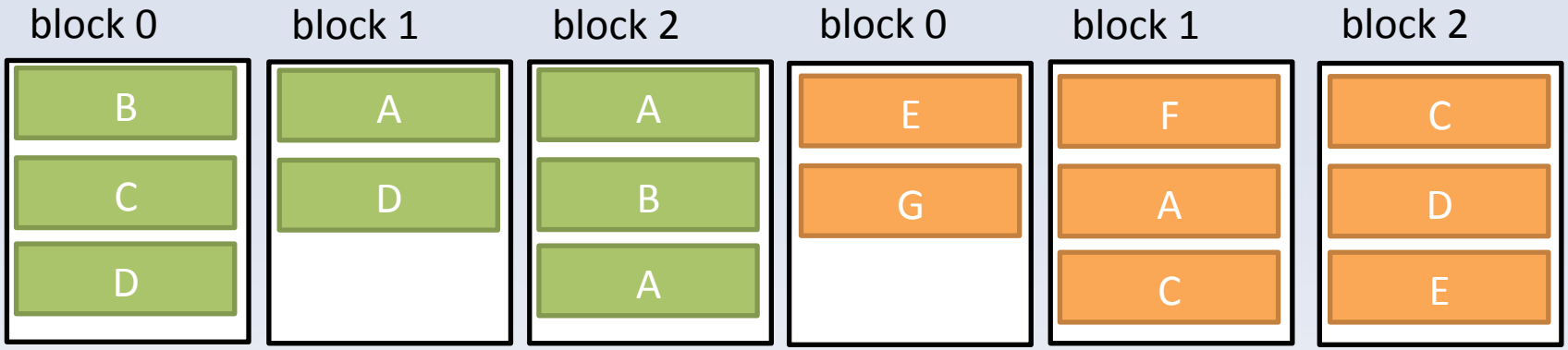
Example R ∩ S

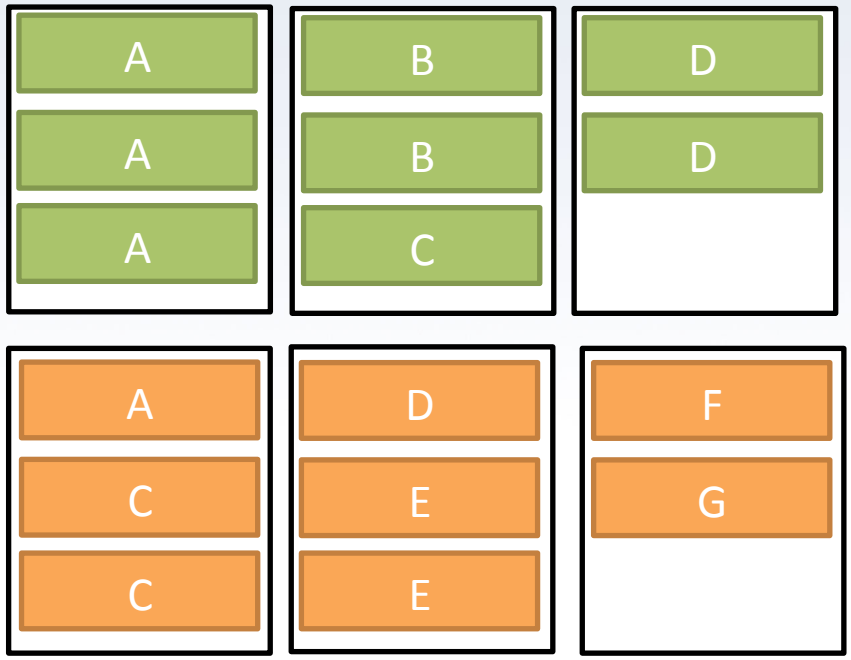

block 0
block 1
block 2
block 0
block 1
block 2
B
C
D
A
D
A
B
A
E
G
F
A
C
C
D
E
E
G
F
A
C
C
D
E
A
A
A
B
B
C
D
D

Example R ∩ S


block 0
B
C
D
block 1
A
D
block 2
A
B
A
block 0
E
G
block 1
F
A
C
block 2
C
D
E
A
C
C
D
E
E
F
G
A
A
A
B
B
C
D
D

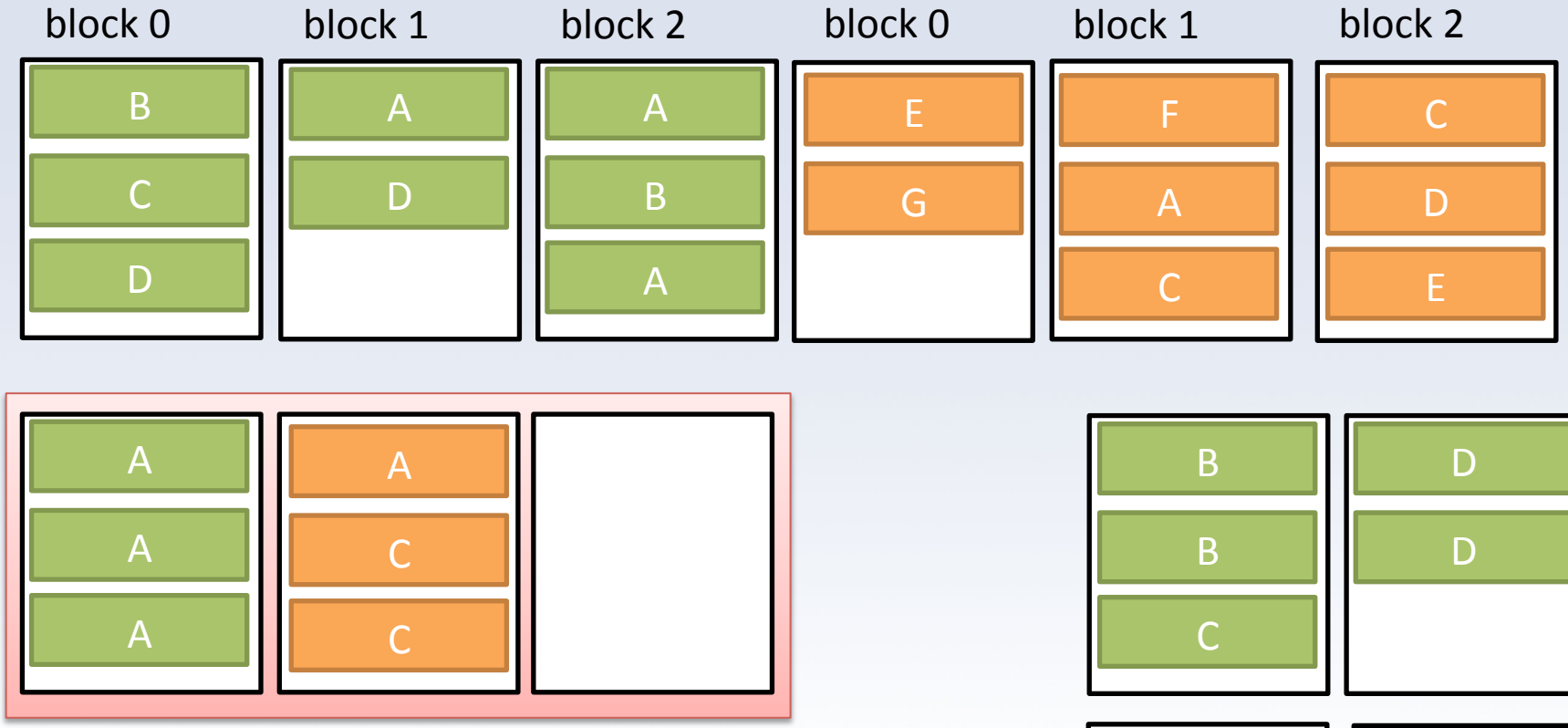
Example R ∩ S



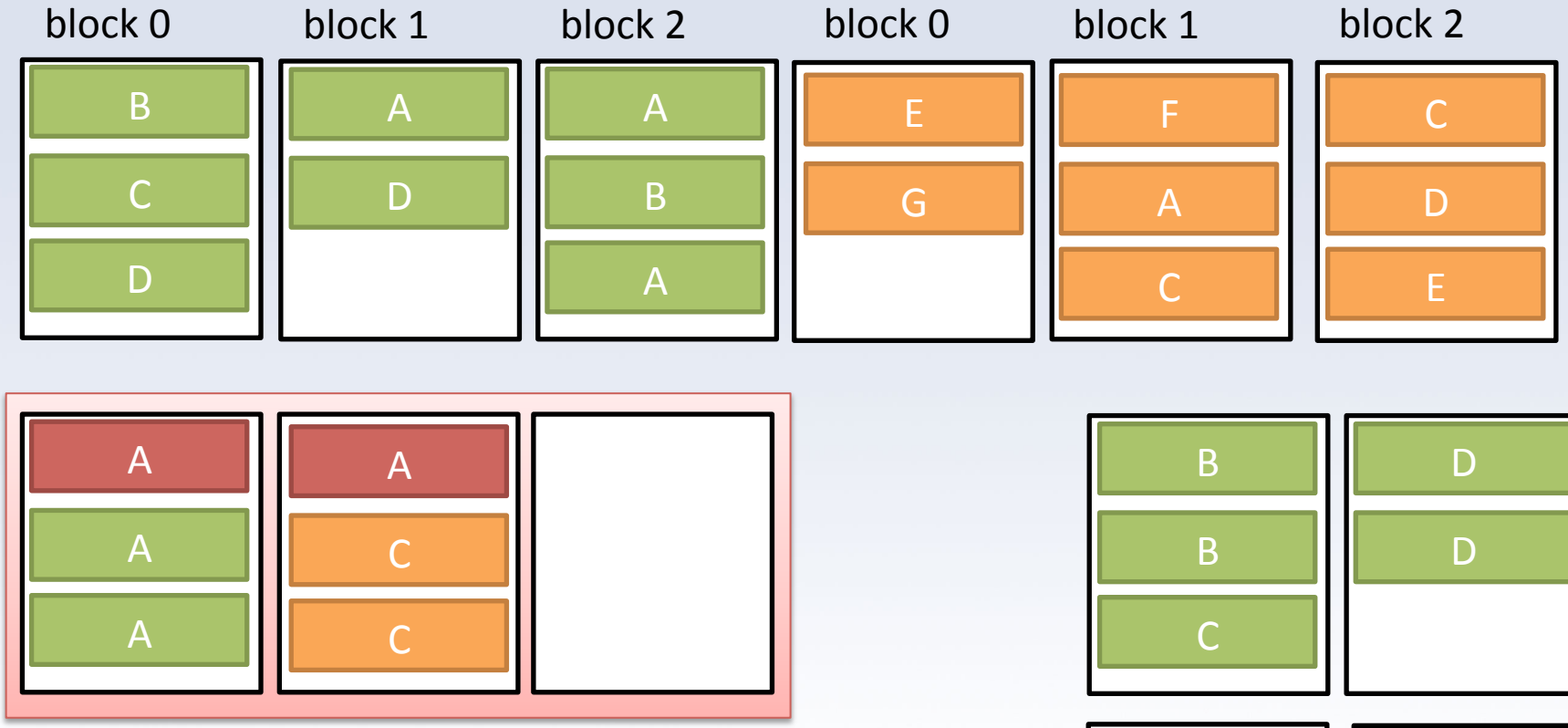


illinois.edu

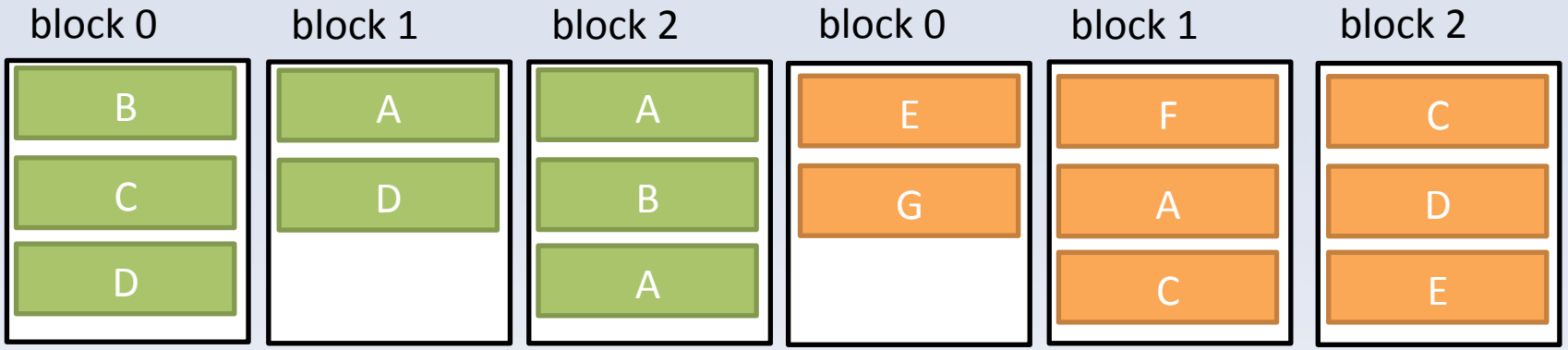
Example R ∩ S

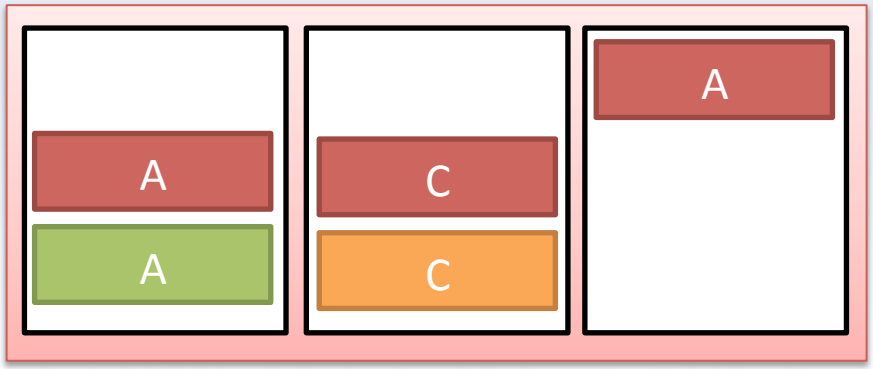

block 0
B
C
D
block 1
A
D
block 2
A
B
A
block 0
E
G
block 1
F
A
C
block 2
C
D
E
A
A
A
A
C
C
B
B
C
D
D
D
E
E
F
G

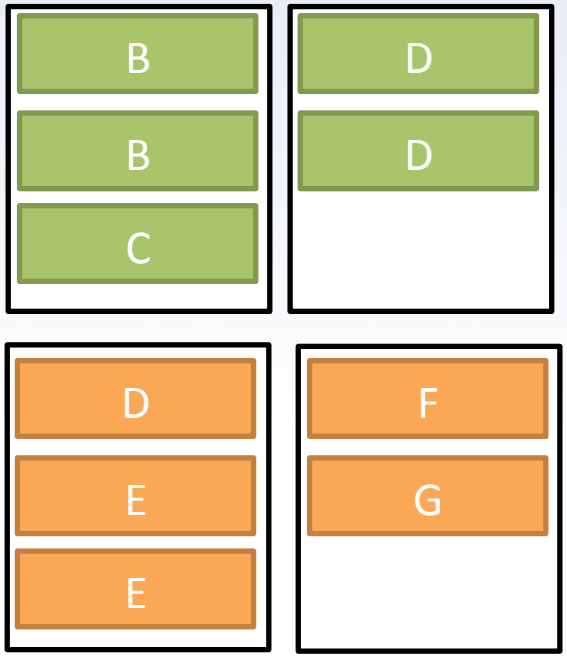
Example R ∩ S


block 0
B
C
D
block 1
A
D
block 2
A
B
A
block 0
E
G
block 1
F
A
C
block 2
C
D
E
A
A
A
A
C
C
B
B
C
D
D
D
E
E
F
G

Example R ∩ S

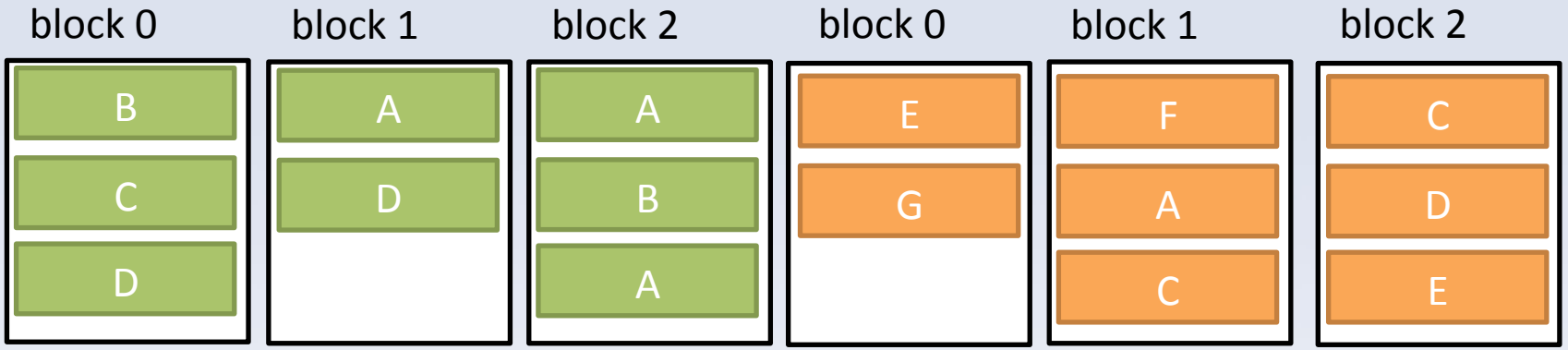


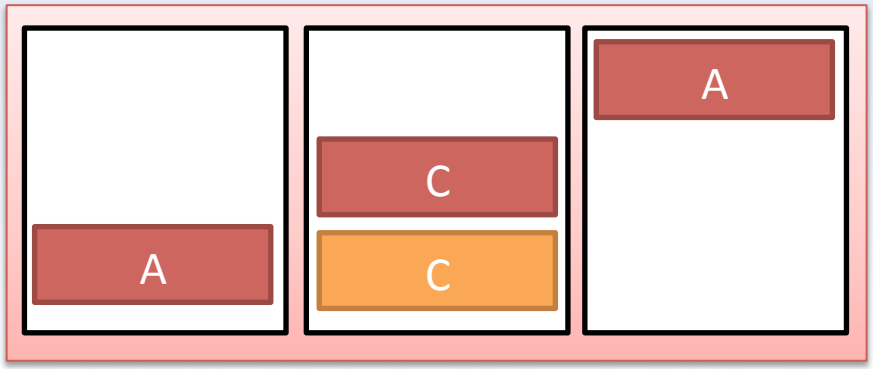


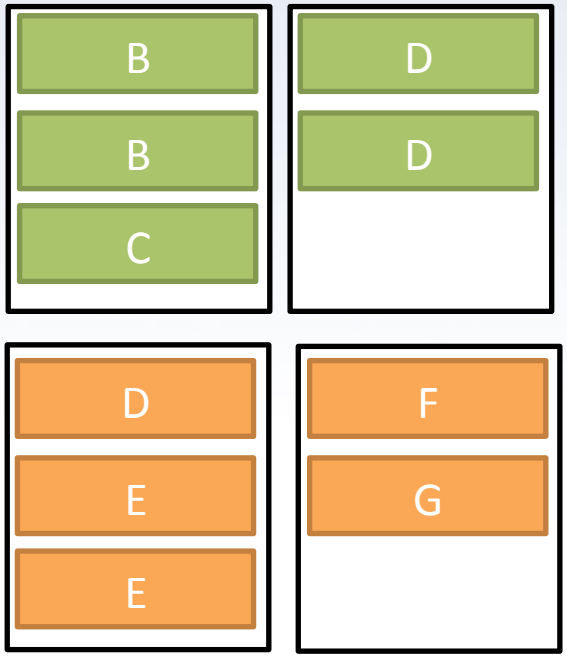


illinois.edu

Example R ∩ S

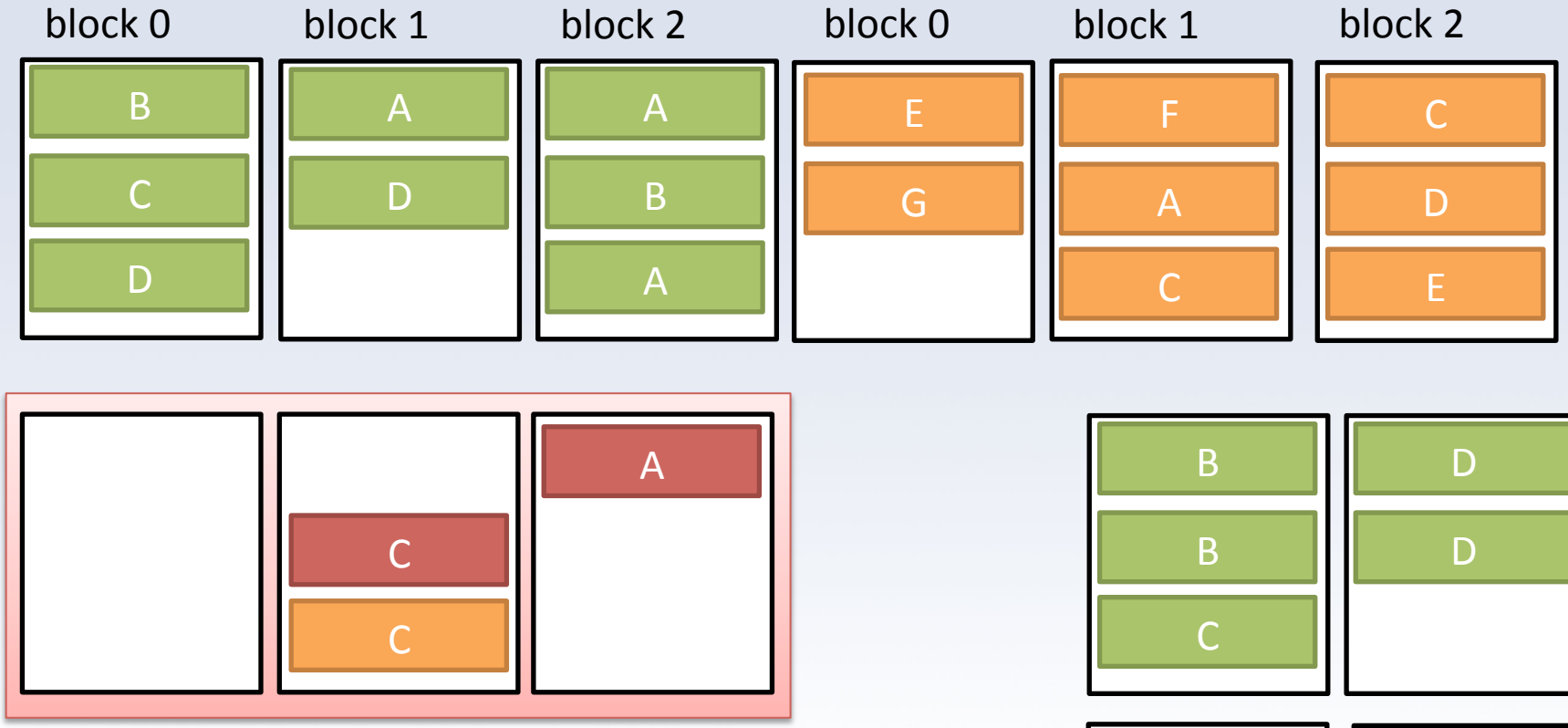




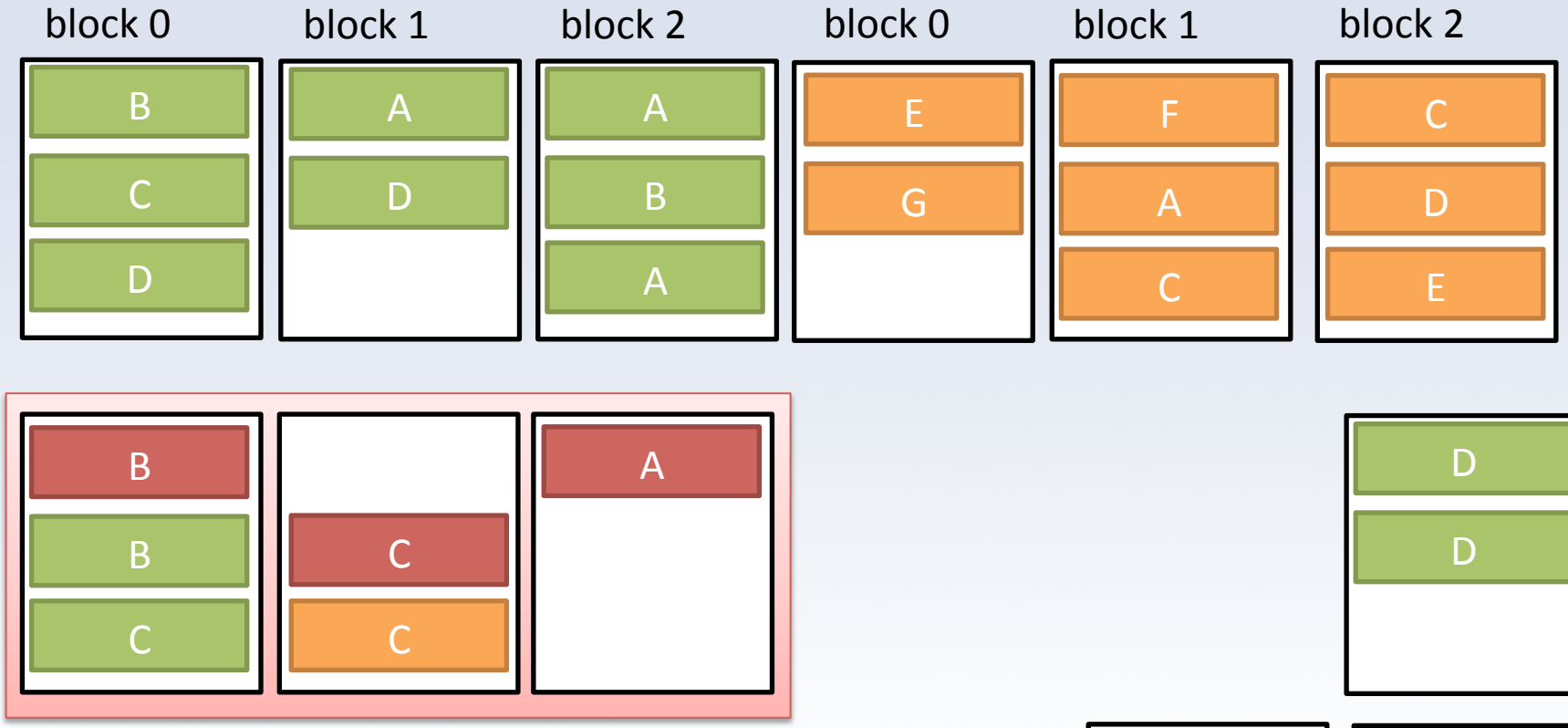


illinois.edu

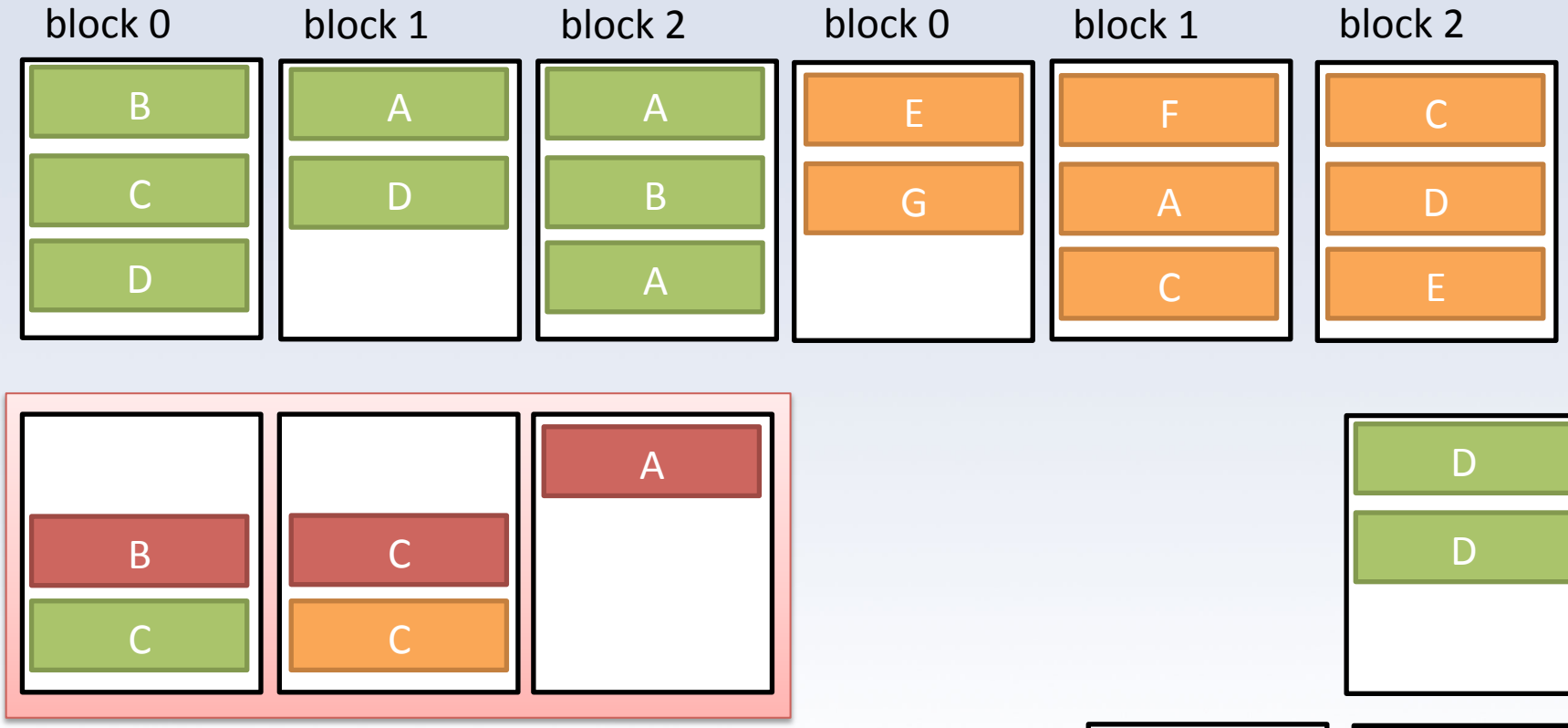
Example R ∩ S


block 0
block 1
block 2
block 0
block 1
block 2
B
C
D
A
D
A
B
A
E
G
F
A
C
C
D
E
C
C
A
B
B
C
D
D
D
E
E
F
G

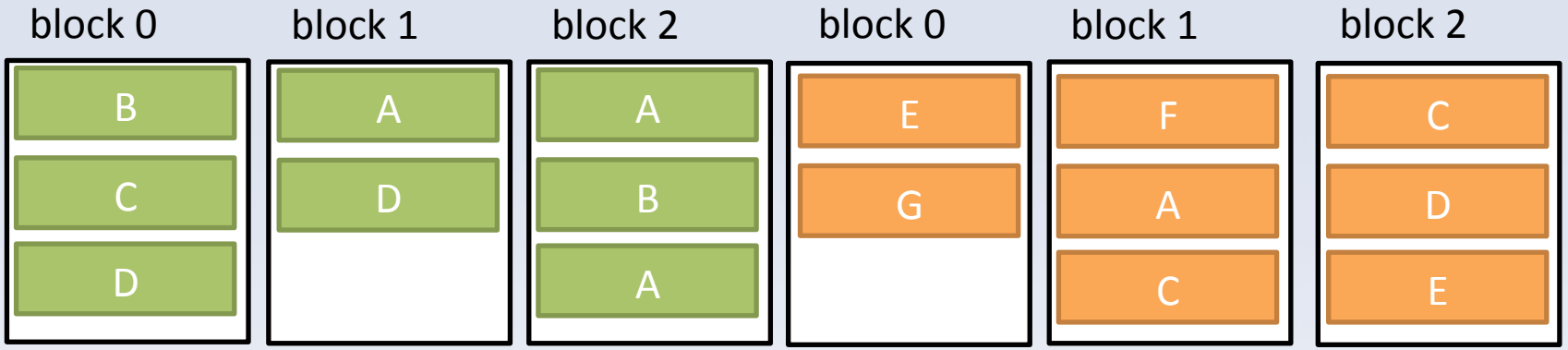
Example R ∩ S

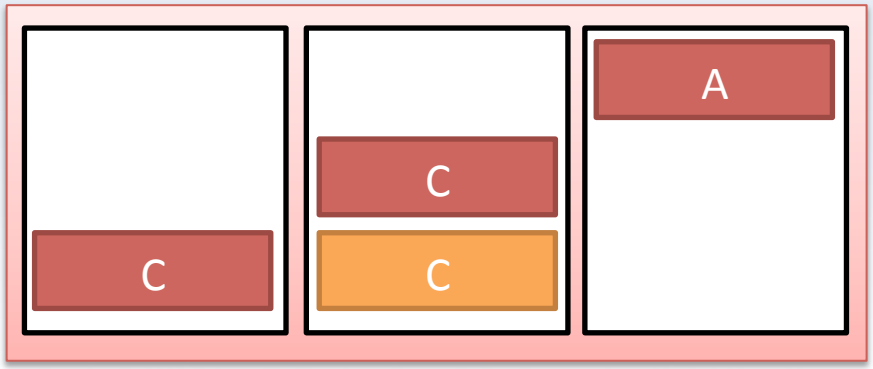

block 0
B
C
D
block 1
A
D
block 2
A
B
A
block 0
E
G
block 1
F
A
C
block 2
C
D
E
B
B
C
C
C
A
D
D
D
E
E
F
G

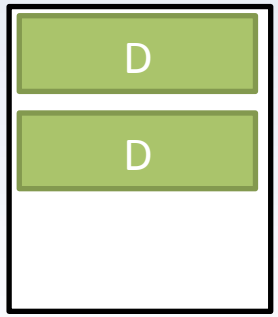
Example R ∩ S

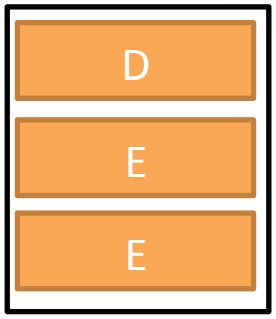

block 0
block 1
block 2
block 0
block 1
block 2
B
C
D
A
D
A
B
A
E
G
F
A
C
C
D
E
B
C
C
C
A
D
D
D
E
E
F
G

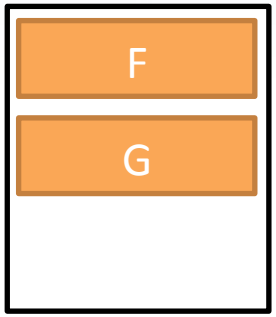
Example R ∩ S


block 0
block 1
block 2
block 0
block 1
block 2
B
C
D
A
D
A
B
A
E
G
F
A
C
C
D
E

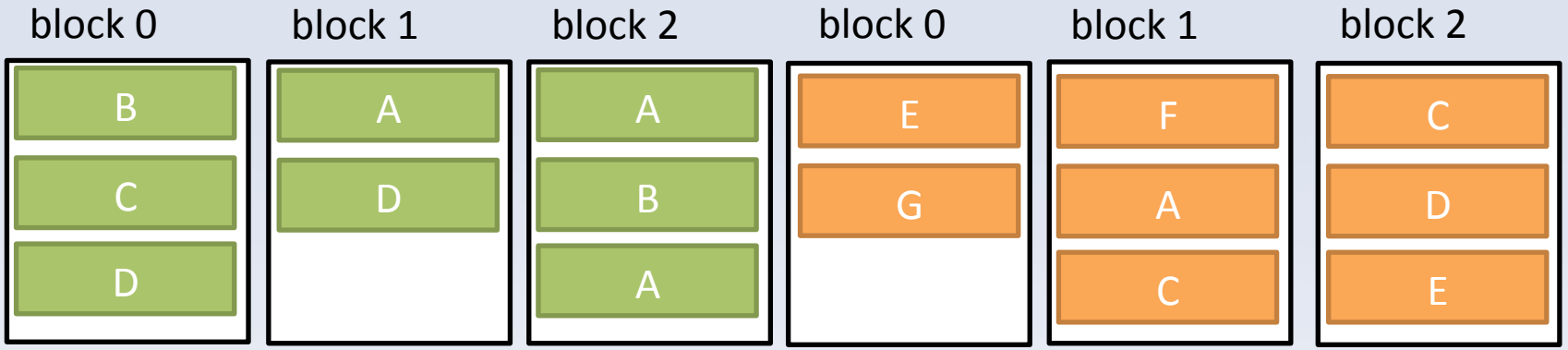

C
C
C
A

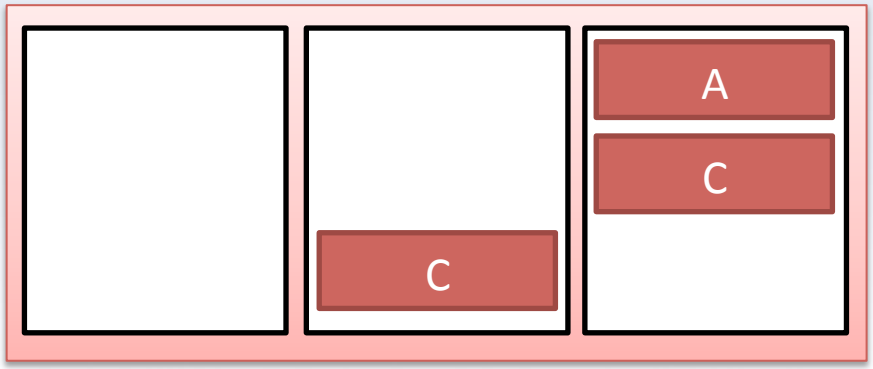

D
D

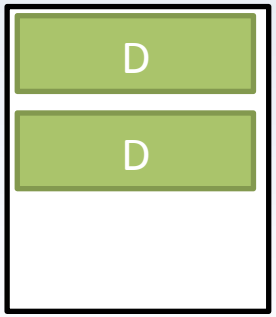

D
E
E

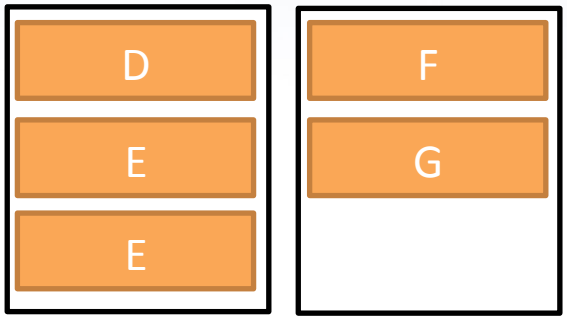

F
G

Example R ∩ S

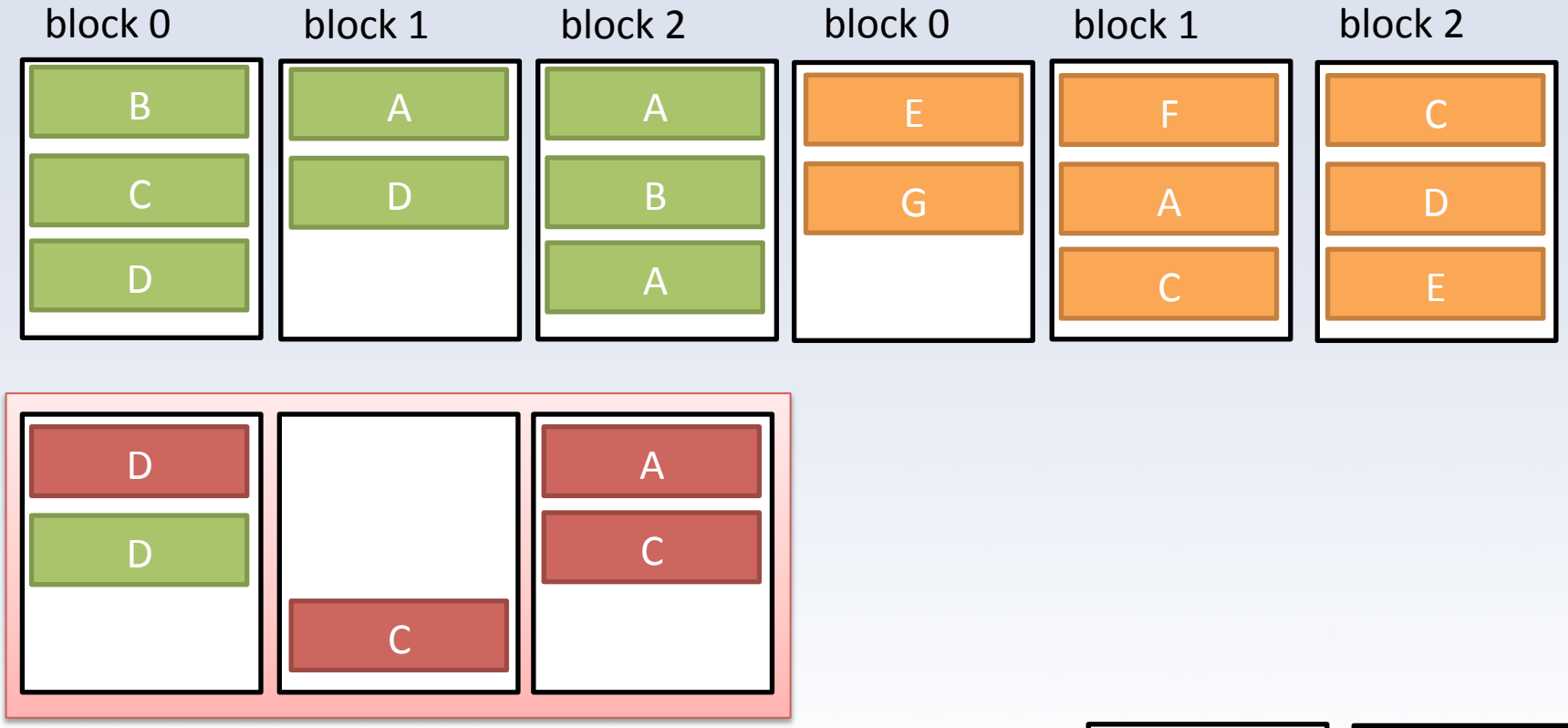

block 0
B
C
D
block 1
A
D
block 2
A
B
A
block 0
E
G
block 1
F
A
C
block 2
C
D
E


A
C
C


D
D

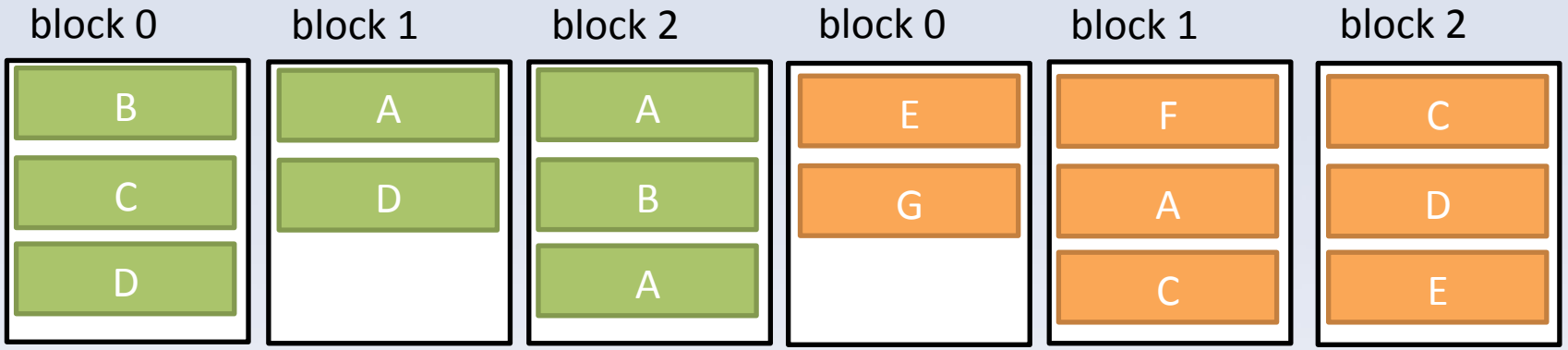

D
E
E
F
G

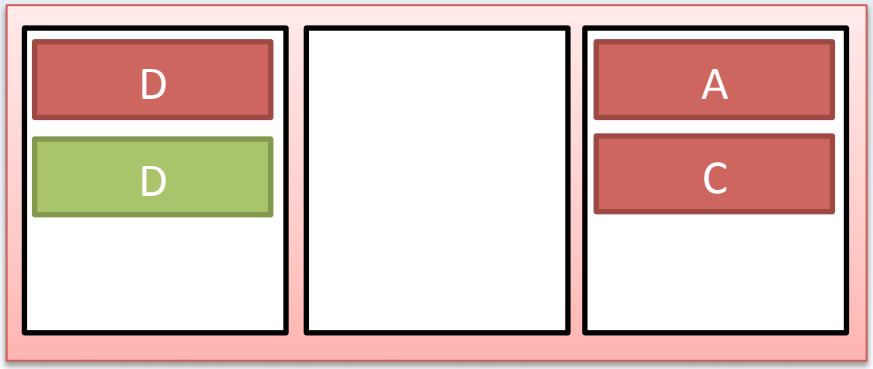
Example R ∩ S

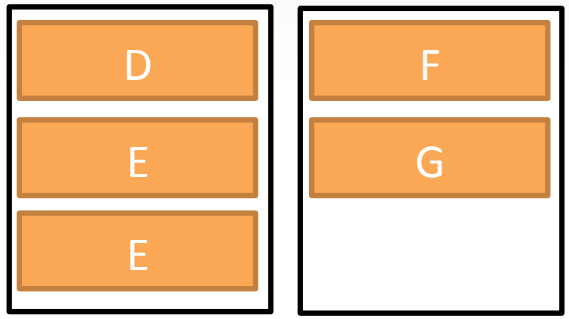


illinois.edu

Example R ∩ S

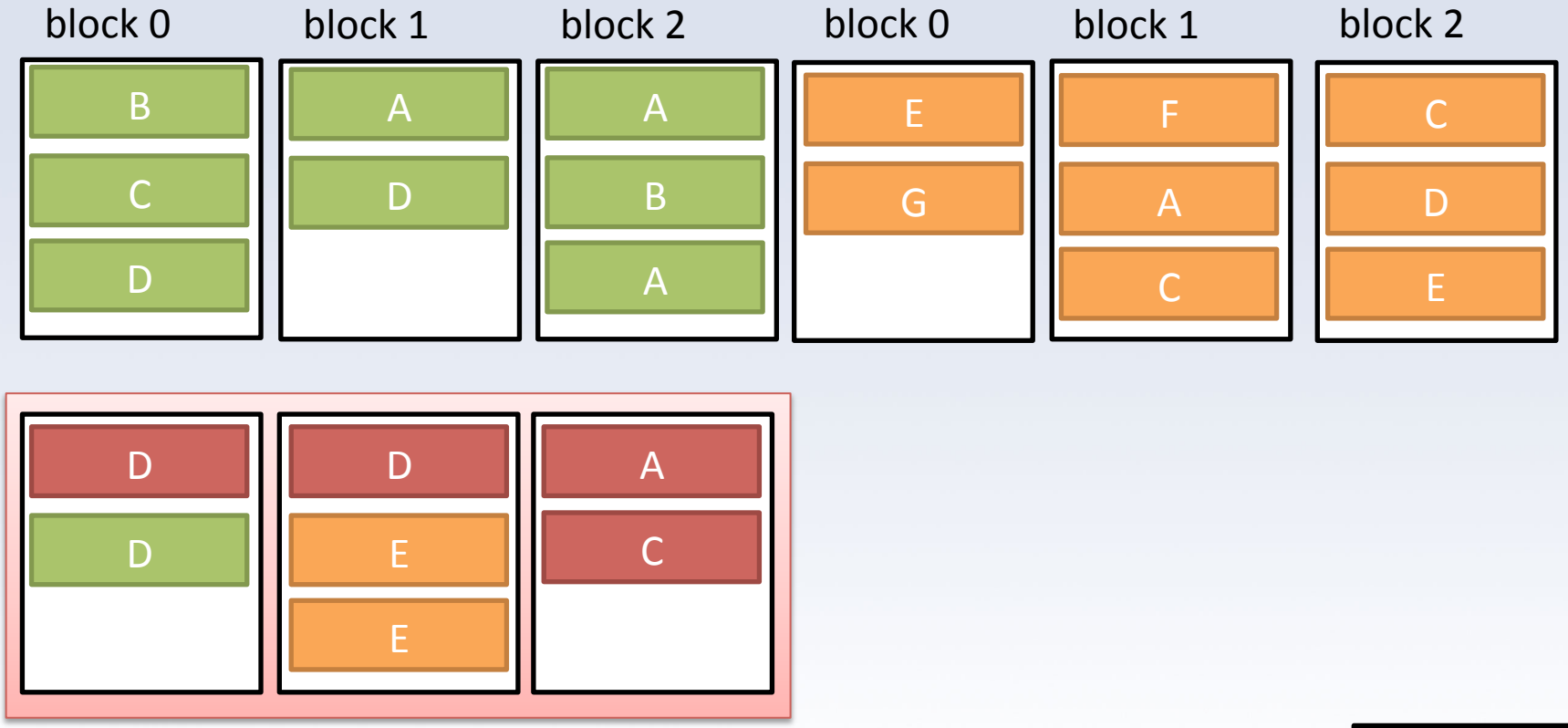




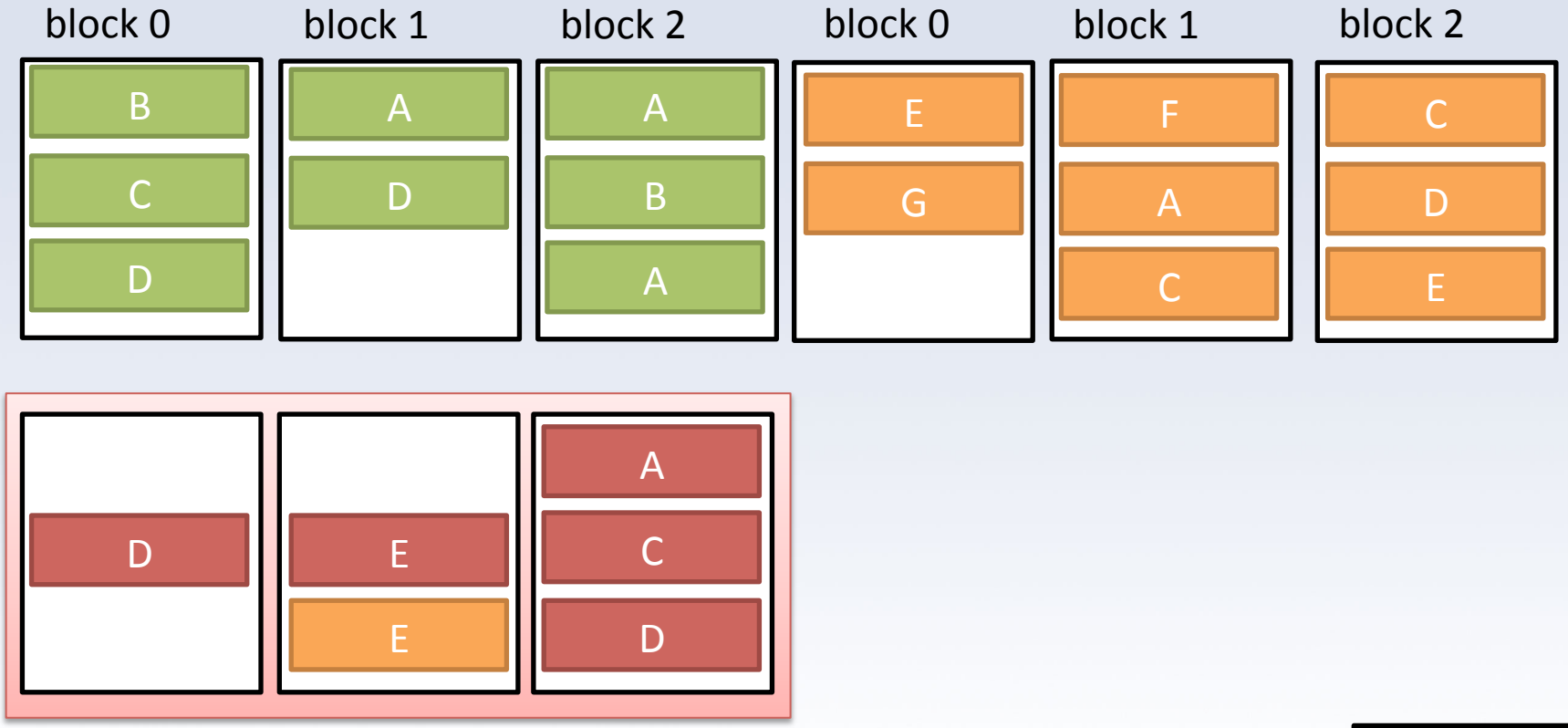


illinois.edu

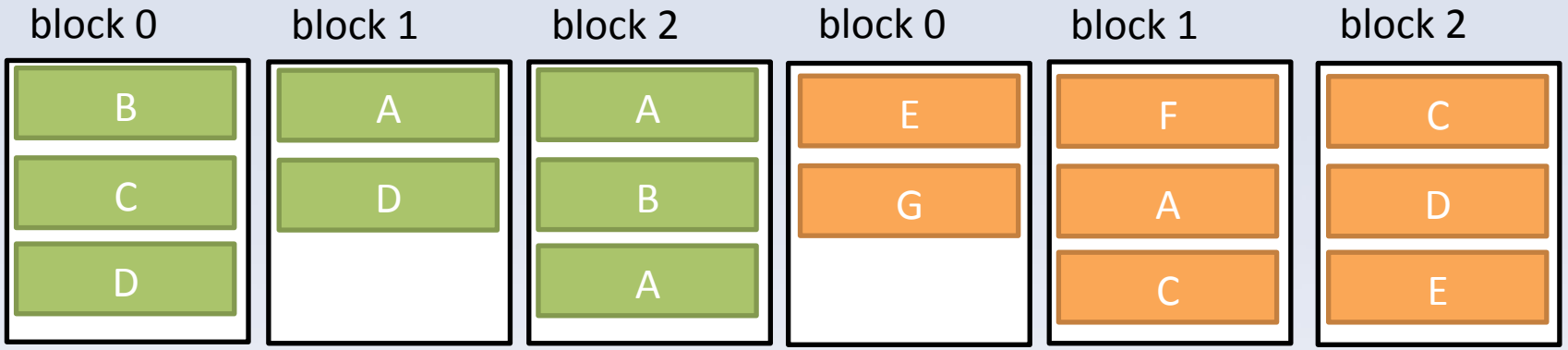
Example R ∩ S

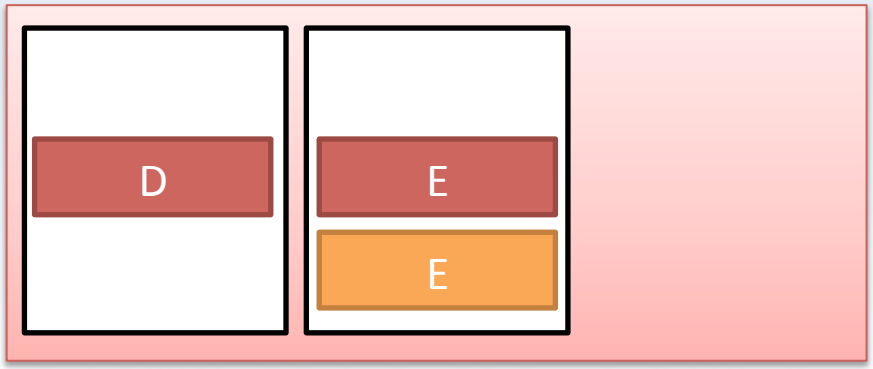

block 0
B
C
D
block 1
A
D
block 2
A
B
A
block 0
E
G
block 1
F
A
C
block 2
C
D
E
D
D
D
D
E
E
A
C
F
G

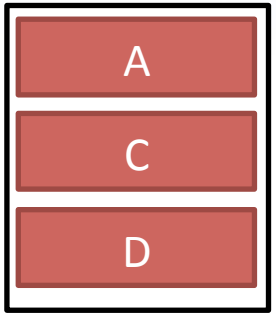
Example R ∩ S

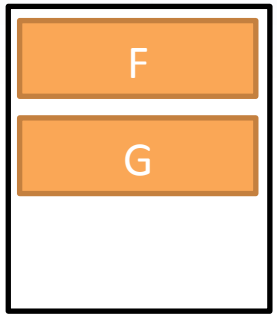

block 0
B
C
D
block 1
A
D
block 2
A
B
A
block 0
E
G
block 1
F
A
C
block 2
C
D
E
D
E
E
A
C
D
F
G

Example R ∩ S

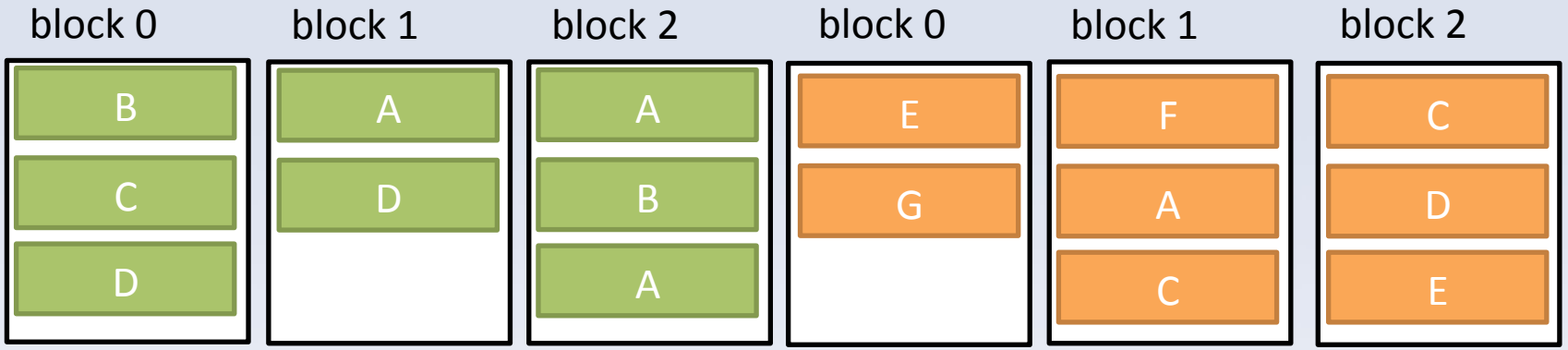

block 0
B
C
D
block 1
A
D
block 2
A
B
A
block 0
E
G
block 1
F
A
C
block 2
C
D
E

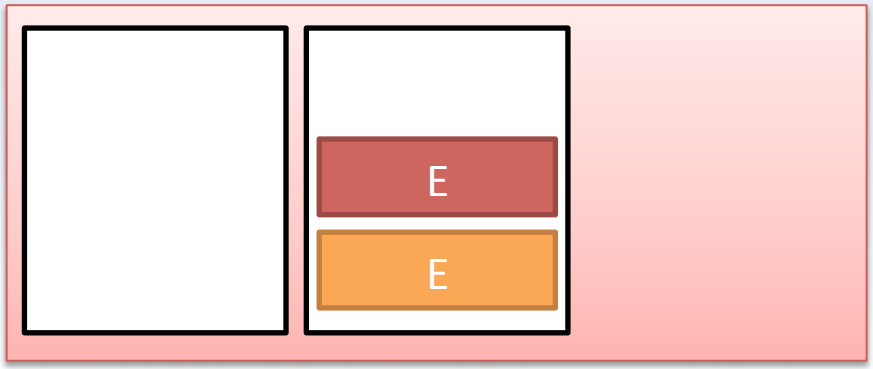

D
E
E

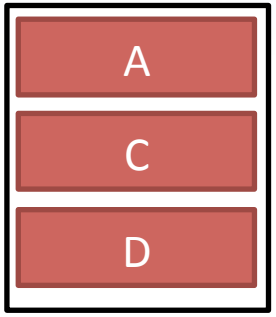

A
C
D

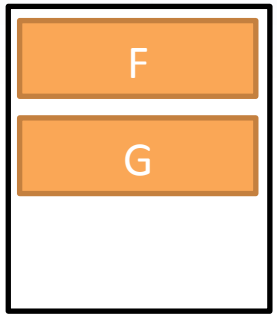

F
G

Example R ∩ S

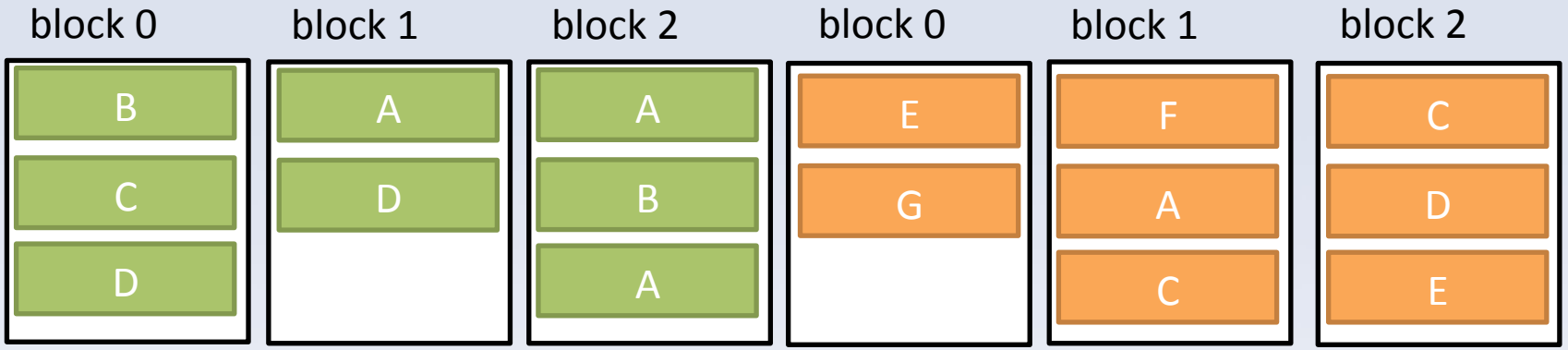

block 0
B
C
D
block 1
A
D
block 2
A
B
A
block 0
E
G
block 1
F
A
C
block 2
C
D
E

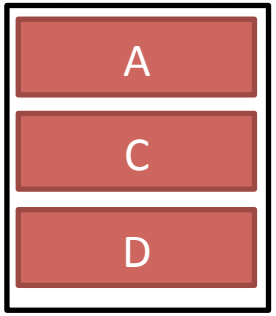

E
E


A
C
D


F
G

Example R ∩ S


block 0
B
C
D
block 1
A
D
block 2
A
B
A
block 0
E
G
block 1
F
A
C
block 2
C
D
E


A
C
D

Sorting-based Set Difference

- We can implement R-S similar to the way we implemented $R \cap S$
- Instead of only outputting when value appears in both relations, only output when it appears in R and not S

Sorting-Based Summary

Operator	M required	I/O Cost
δ, γ	√B	3B
∪ , ∩ , -	√(B(R)+B(S))	3(B(R)+B(S))

Next time…

- Use TPMMS to implement 2 pass algorithms for ⋈ operation
- Use *hashing* to implement 2 pass algorithms for operators
- Take advantage of indexes to implement algorithms for operators