

*Collection-focused*  
*Parallelism*

---



*Micah J Best* | **UBC**  
Nicholas Vining | **UBC**  
Daniel Jacobsen | **Gaslamp Games**  
Alexandra Fedorova | **SFU**



# Clockwork Empires

---

# Clockwork Empires



# Mental Models

---

Argument

Realization

Experience

# Mental Models

---

Argument

Realization

Experience

# Mental Models

---

Argument

Realization

Experience

# Mental Models

---

Argument

Realization

Experience

# Mental Models

---

Argument

Realization

Experience



# Mental Models

---

Argument

Realization

Sub-collections are a good candidate for  
the **basic unit of parallelism**

Experience



# Mental Models

---



# Mental Models

---

How do we think about parallelism?



# Mental Models

---

How do we think about parallelism?



# Mental Models

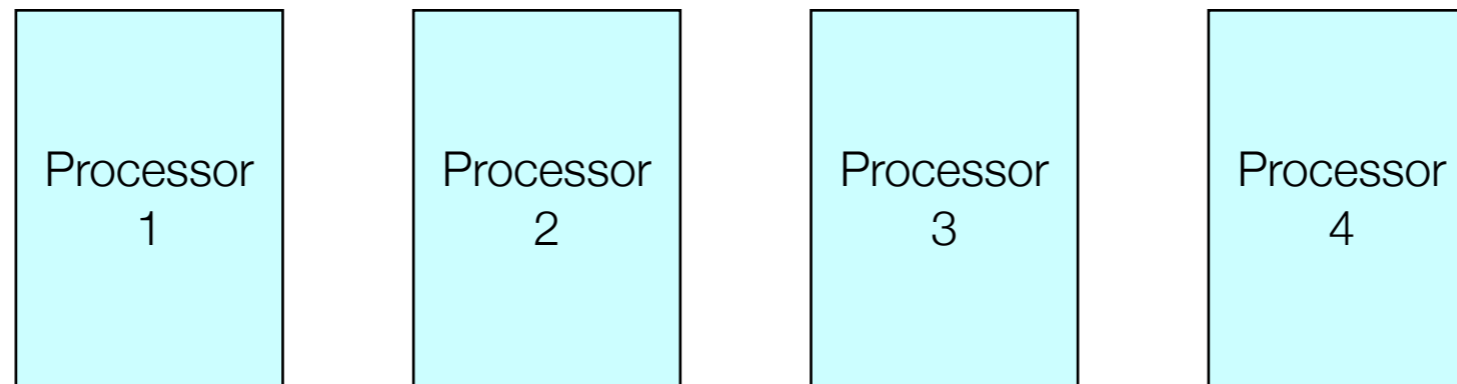
---

## The “Undergrad Model”

# Mental Models

---

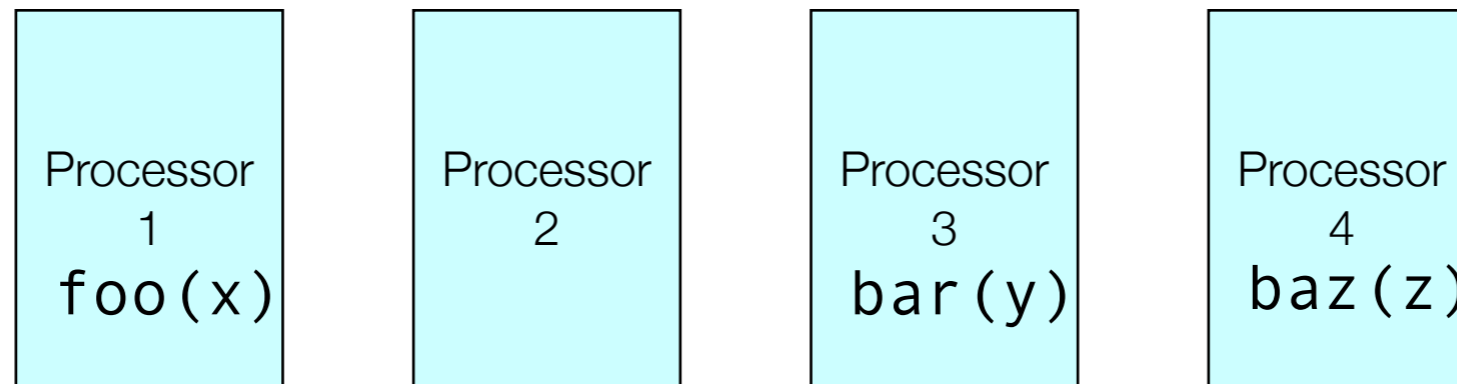
## The “Undergrad Model”



```
int main(int argc, char **argv)
{
    foo(x)
    bar(y)
    baz(z)
}
```

# Mental Models

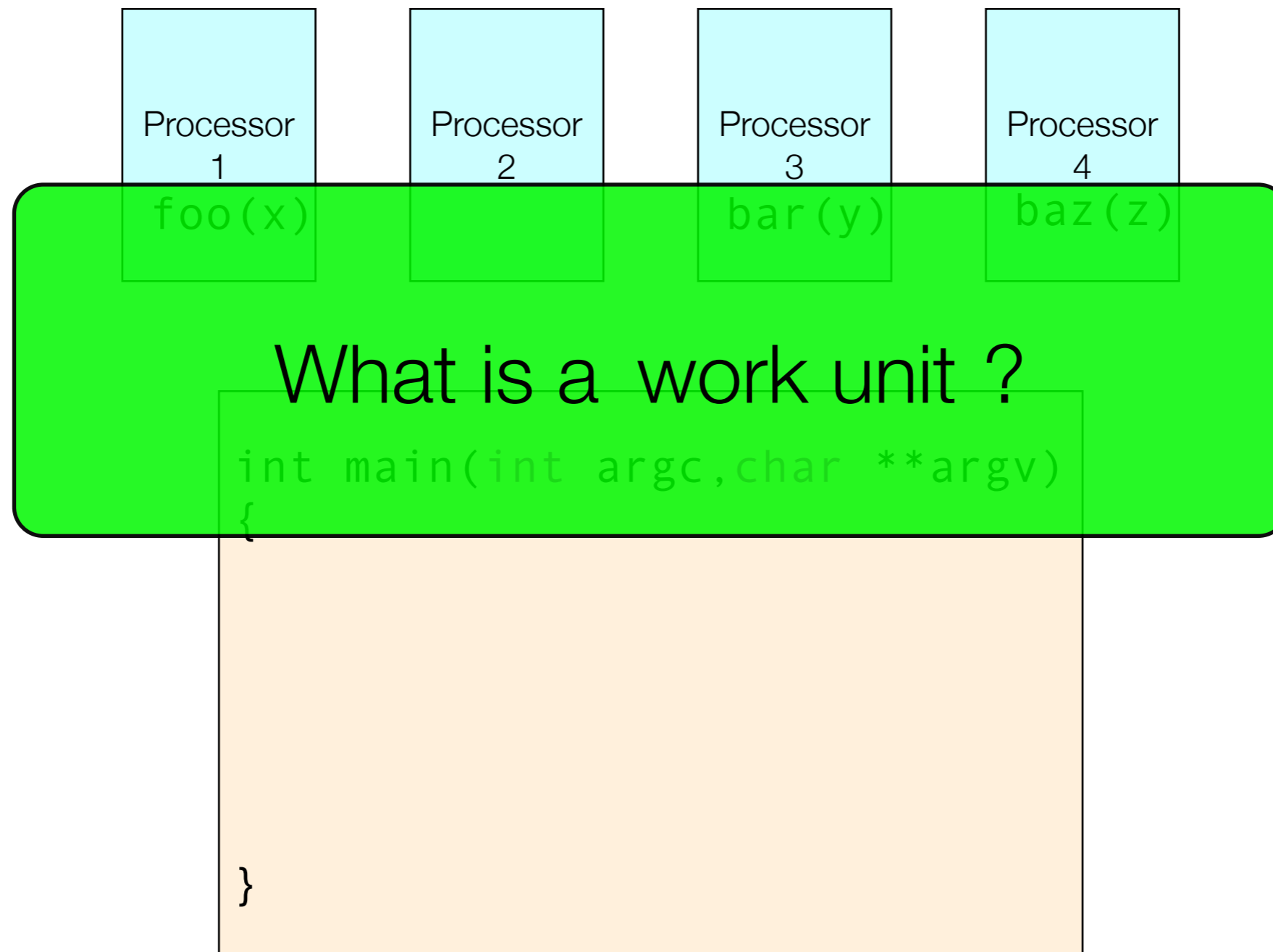
## The “Undergrad Model”



```
int main(int argc, char **argv)
{
}
}
```

# Mental Models

## The “Undergrad Model”







# Mental Models

---



# Mental Models

---

L1 Cache hit: ~**1-3** cycles

L2 Cache hit: **10s** of cycles

Main Memory Access: **100s** of cycles

# Mental Models

---

L1 Cache hit: ~**1-3** cycles

L2 Cache hit: **10s** of cycles

Main Memory Access: **100s** of cycles

Cloud Access: Many many many  
many many cycles

# Mental Models

---

deadlock

L1 Cache hit: ~**1-3** cycles

livelock

L2 Cache hit: **10s** of cycles

load  
balancing

starvation

Main Memory Access: **100s** of cycles

state conflicts

Cloud Access: Many many many  
many many cycles

# Mental Models

deadlock

L1 Cache hit: ~**1-3** cycles

livelock

L2 Cache hit: **10s** of cycles

load  
balancing

starvation

Main Memory Access: **100s** of cycles

state conflicts

Cloud Access: Many many many

many many cycles

branch  
misprediction



# Schedule Data, Not Code

---



# Schedule Data, Not Code

---

What if we pick up the other end of the stick?



# Schedule Data, Not Code

---

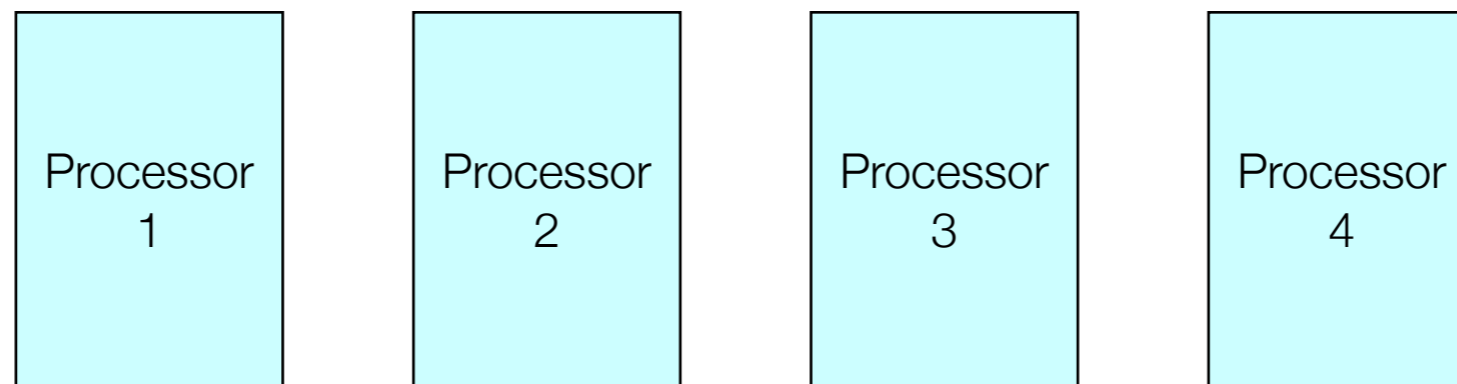
What if we pick up the other end of the stick?



# Schedule Data, Not Code

---

What if we pick up the other end of the stick?

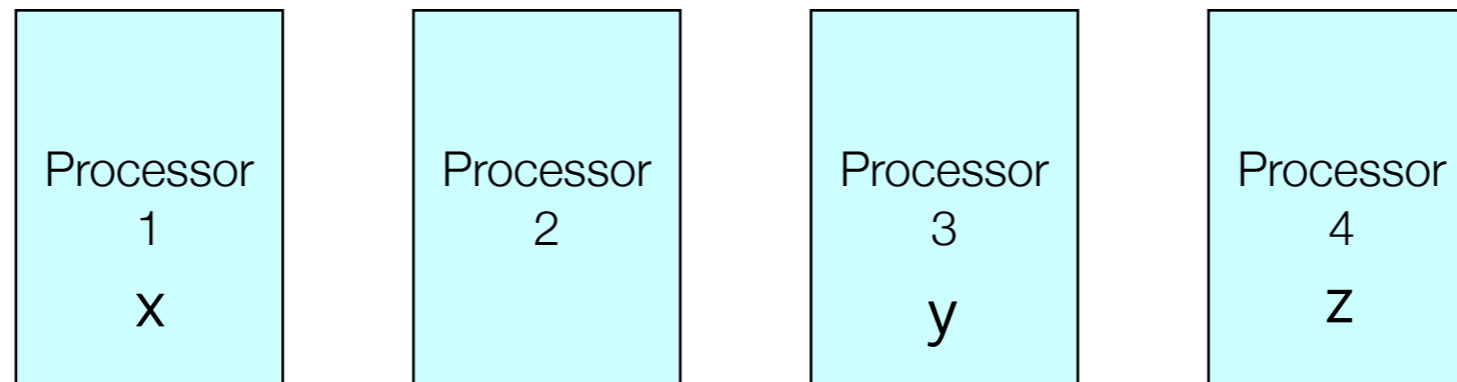


```
int main(int argc, char **argv)
{
    foo(x)
    bar(y)
    baz(z)
}
```

# Schedule Data, Not Code

---

## Schedule Data, Not Code



```
int main(int argc, char **argv)
{
}

```



# Data is in Collections

---

Code is static

Data is dynamic



# Data is in Collections

---

Code is static

Data is dynamic

But data never travels alone.

# Data is in Collections

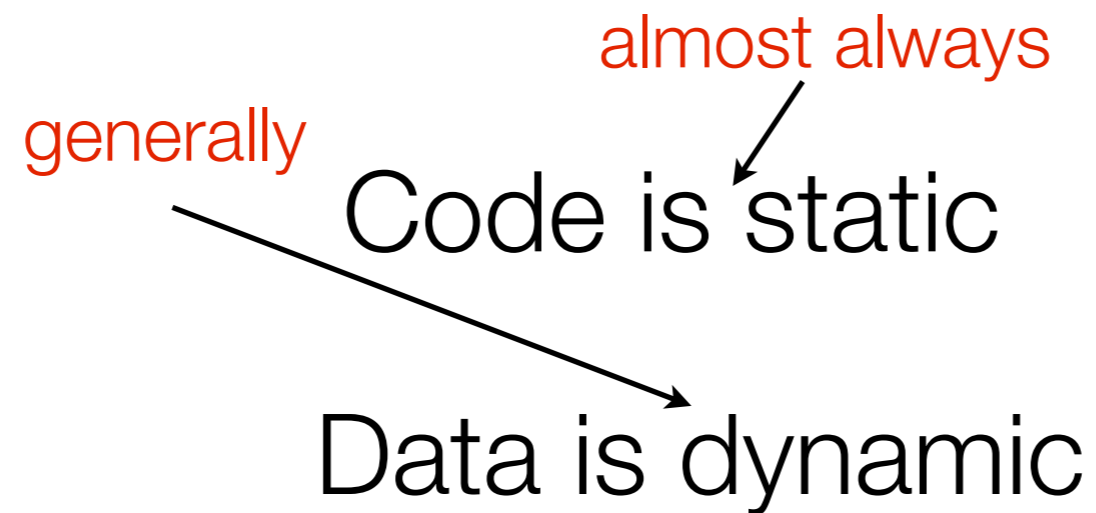
---

Code is static

almost always

generally

Data is dynamic



The diagram consists of two lines of text. The top line is 'Code is static'. Above it, the words 'almost always' are written in red, with a black arrow pointing down to the word 'static'. To the left of 'Code is static', the word 'generally' is written in red, with a black arrow pointing down and to the right towards the text 'Data is dynamic' on the line below.

But data ~~never~~ travels alone.

rarely



The text 'But data never travels alone.' is written in black. The word 'never' is crossed out with a red diagonal line. Above the word 'never', the word 'rarely' is written in red.

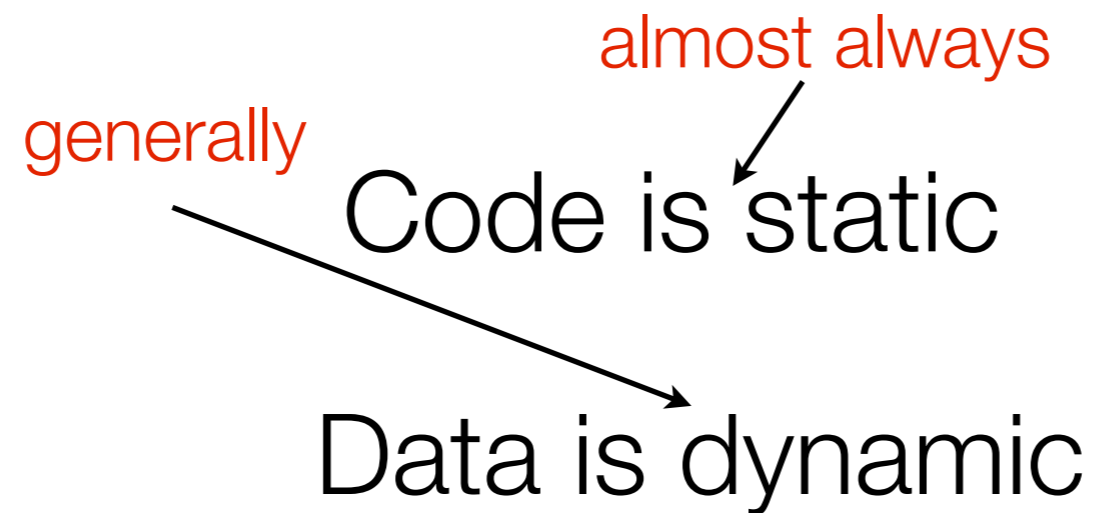
# Data is in Collections

---

Code is static  
Data is dynamic

almost always

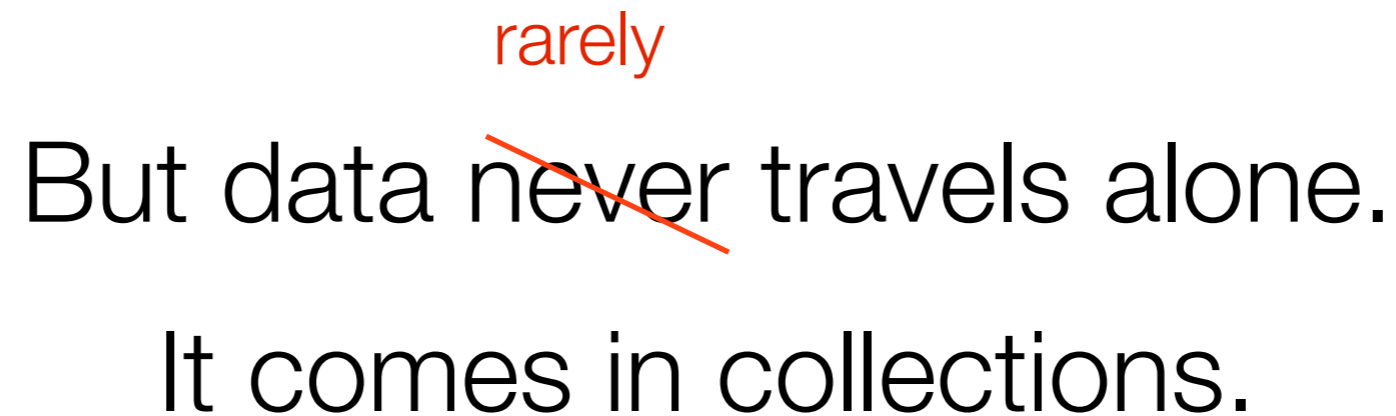
generally



The diagram consists of two lines of text: 'Code is static' on top and 'Data is dynamic' on the bottom. A black arrow points from 'Code is static' down to 'Data is dynamic'. To the left of 'Code is static' is the word 'generally' in red. Above 'Code is static' is the phrase 'almost always' in red, with a black arrow pointing down to the word 'static'.

But data ~~never~~ travels alone.  
It comes in collections.

rarely



The text 'But data never travels alone.' has the word 'never' crossed out with a red diagonal line. Above the word 'never' is the word 'rarely' in red.

# Data is in Collections

---

Code is static  
Data is dynamic

Diagram illustrating the relationship between code and data:

- Code is static (annotated with "almost always")
- Data is dynamic (annotated with "generally")
- An arrow points from "Code is static" to "Data is dynamic", indicating that data is dynamic because code is static.

But data ~~never~~ travels alone.

It comes in collections.

We rarely use a whole collection at once.



# Collections

---





# Collections

---

What is a work unit?

# Collections

---

What is a work unit?

The smallest **set of subcollections** needed for processing in making **forward progress** in the application.

# Mental Models

---

Argument

Realization

Experience

# Mental Models

---

Argument

Realization

Experience



# Realization Problems

---



# Realization Problems

---

How do we efficiently deal with sub-collections?



# Realization Problems

---

How do we efficiently deal with sub-collections?

How do we structure programs?



# Realization Problems

---

How do we efficiently deal with sub-collections?

How do we structure programs?

How do we derive schedules?





# Synchronization via Scheduling (SvS)

---

Basic Idea:

Basic Method:



# Synchronization via Scheduling (SvS)

---

Basic Idea:

Know what data a task is going to access before it executes and use this information to make scheduling decisions.

Basic Method:

# Synchronization via Scheduling (SvS)

---

Basic Idea:

Know what data a task is going to access before it executes and use this information to make scheduling decisions.

Basic Method:

Derive a compact representation (a single bit string) of the ‘space’ of potential access for quick comparisons during scheduling.



# Software Patterns (IMR)

---

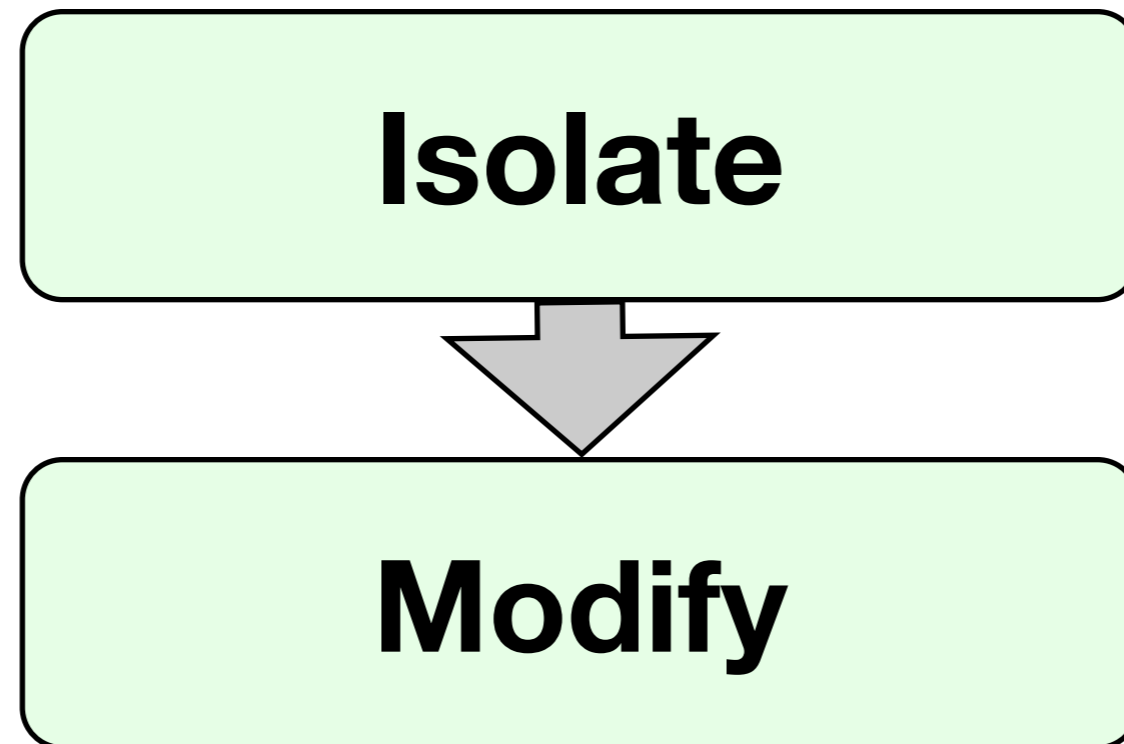
# Software Patterns (IMR)

---

**Isolate**

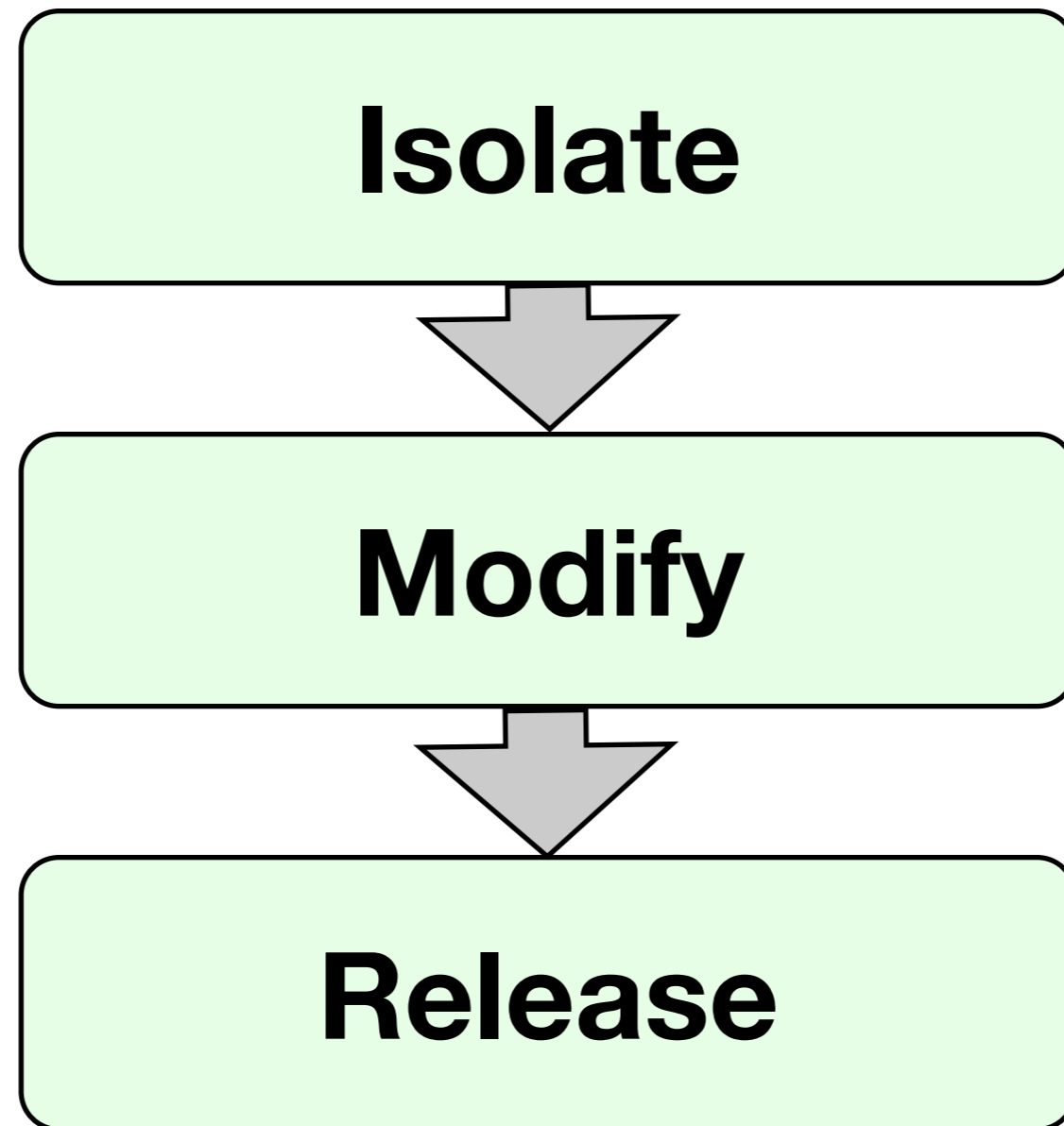
# Software Patterns (IMR)

---



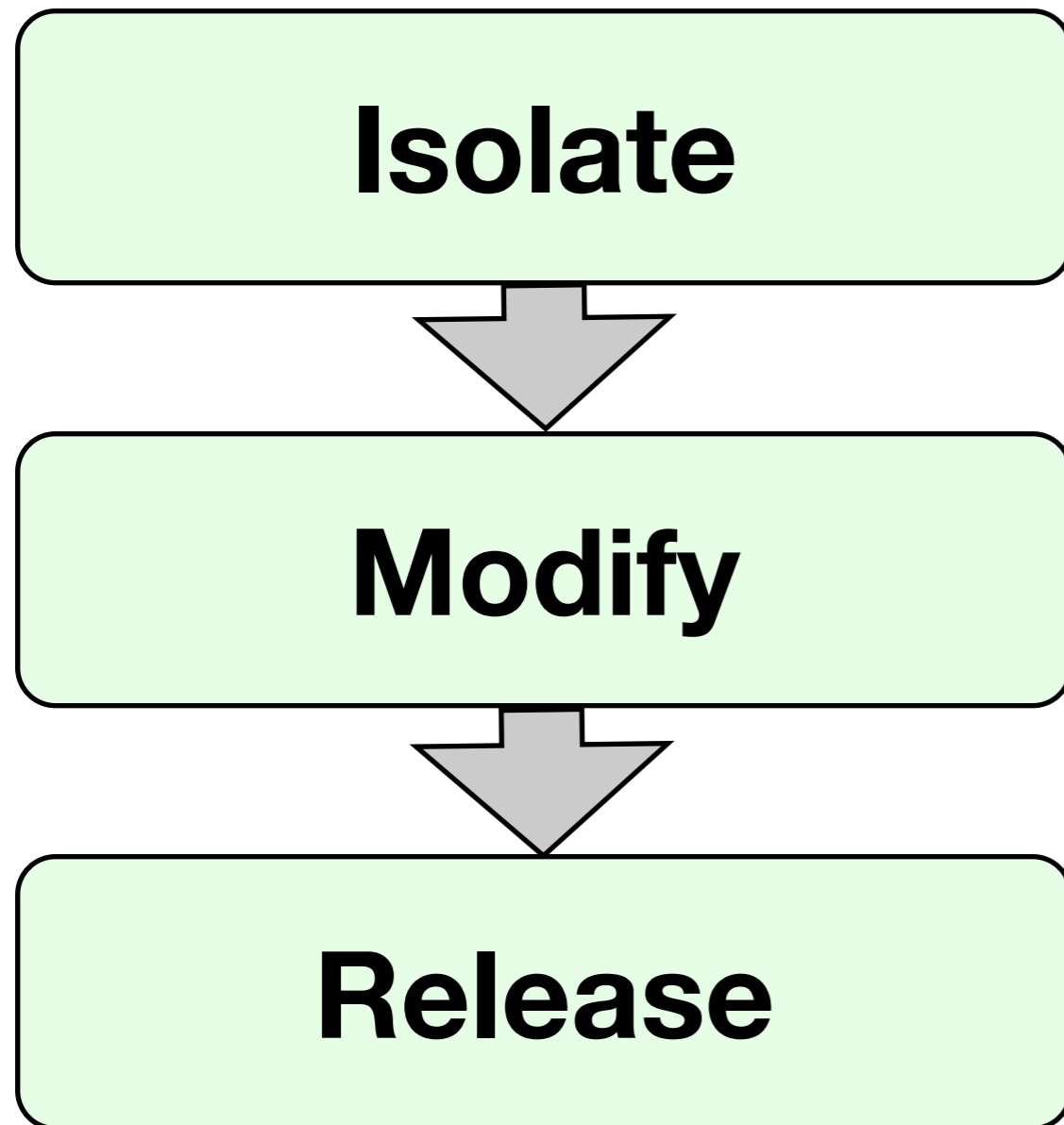
# Software Patterns (IMR)

---



# Software Patterns (IMR)

---

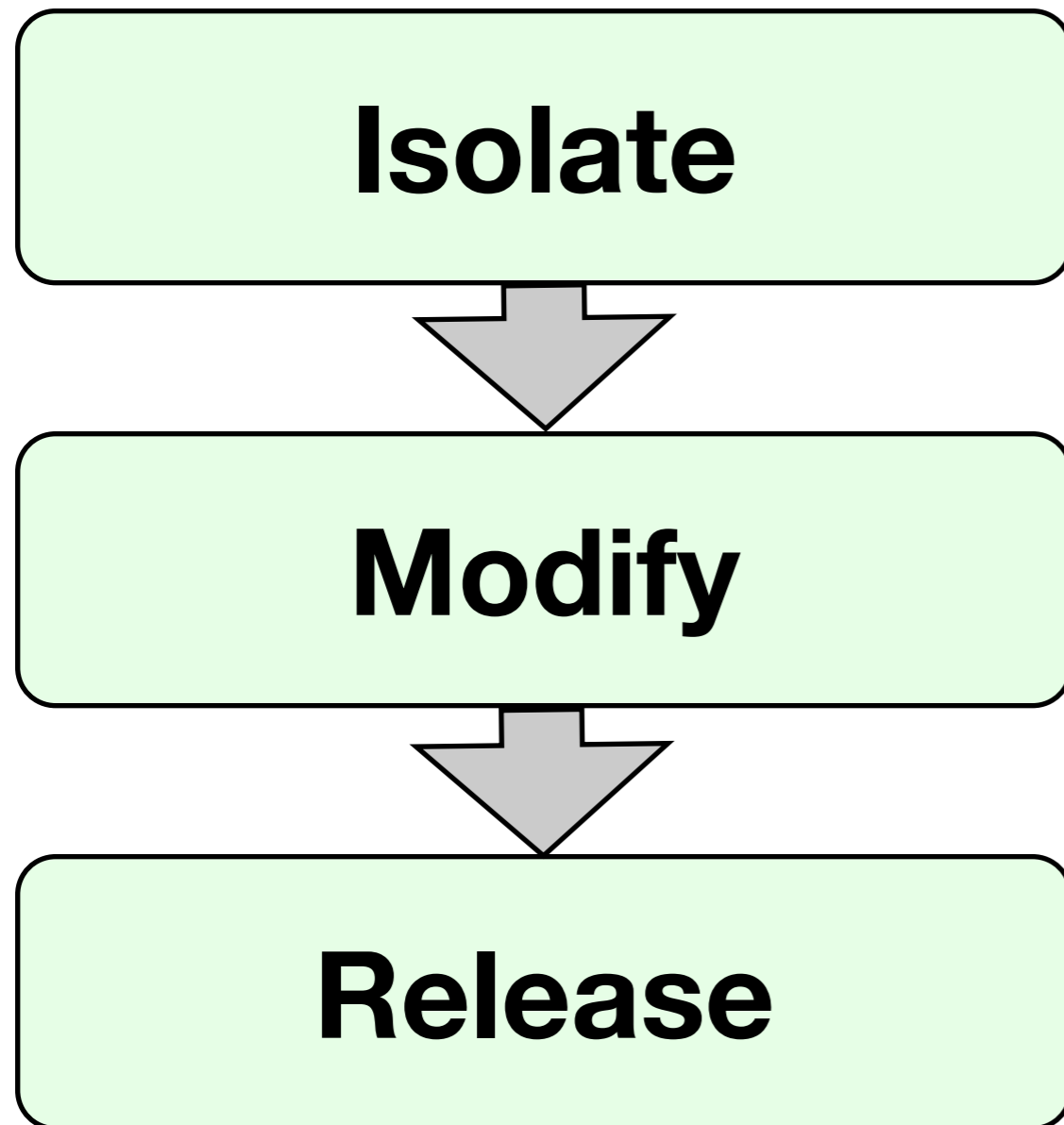




# Software Patterns (IMR)

---

## Stencil Patterns



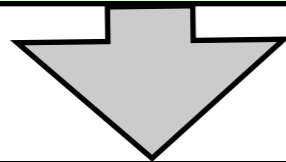
# Software Patterns (IMR)

---

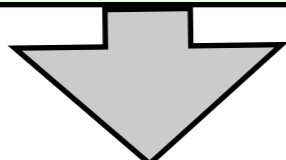
Stencil Patterns

List modification

**Isolate**



**Modify**



**Release**

# Software Patterns (IMR)

---

Stencil Patterns

List modification

Tree Modification

**Isolate**



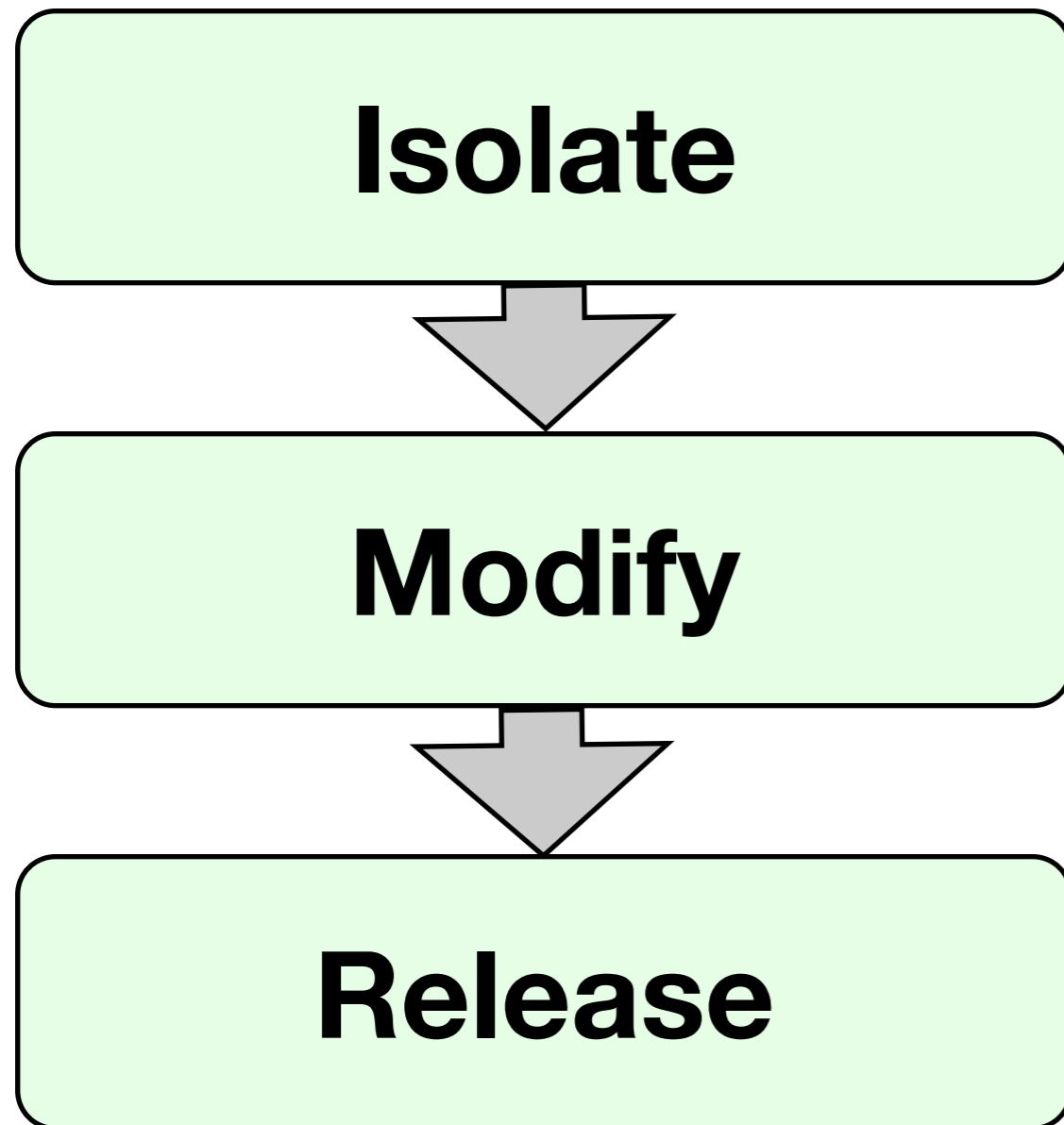
**Modify**



**Release**

# Software Patterns (IMR)

---



Stencil Patterns

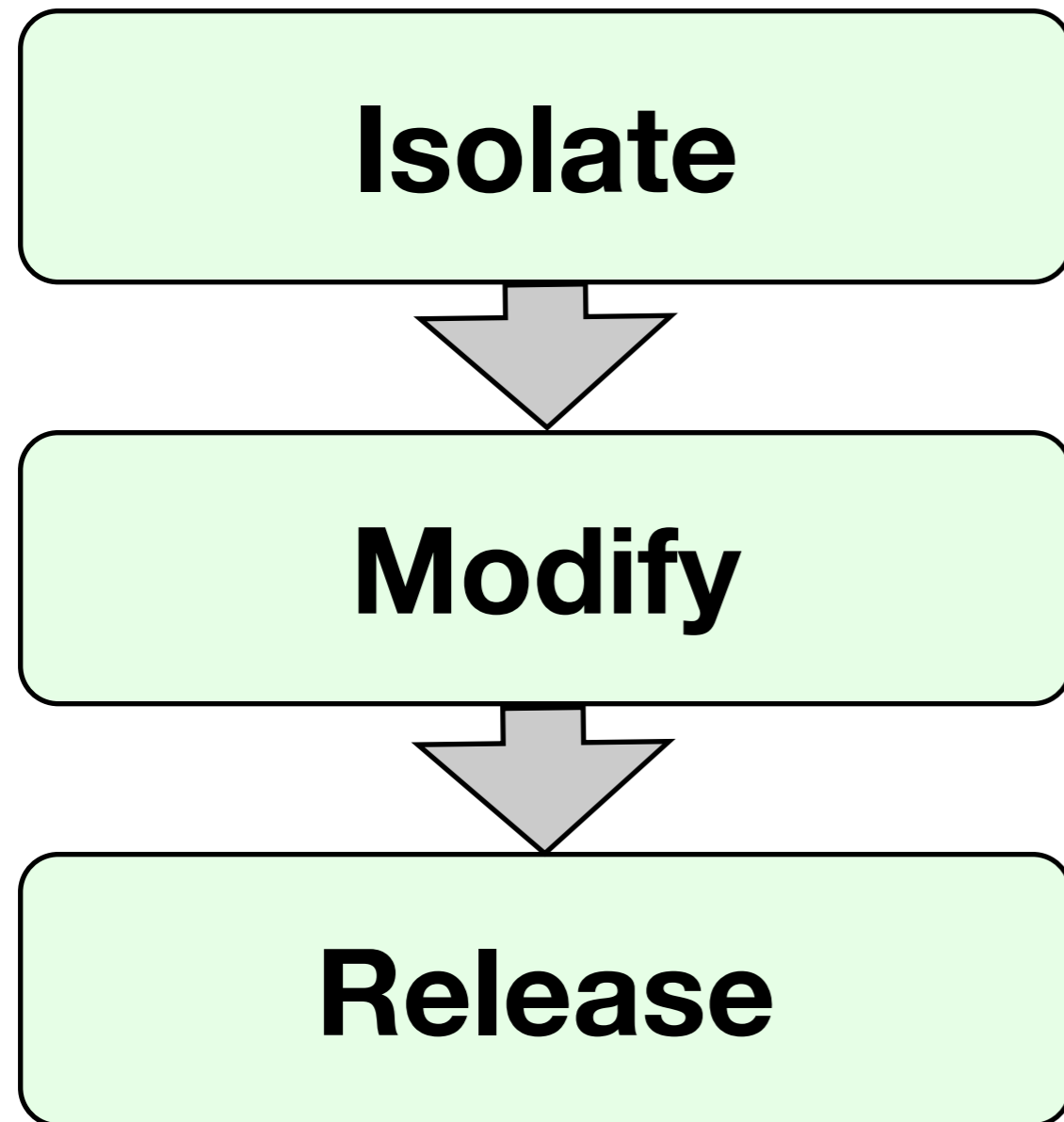
List modification

Tree Modification

Graph Modification

# Software Patterns (IMR)

---



Stencil Patterns

List modification

Tree Modification

Graph Modification

... more



# Programming Support

---



# Programming Support

---

‘First class’ collections



# Programming Support

---

'First class' collections

Actor Model





# Programming Support

---

‘First class’ collections

Actor Model + Messages



# Programming Support

---

‘First class’ collections

Actor Model + Messages + Queries

# Mental Models

---

Argument

Realization

Experience

# Mental Models

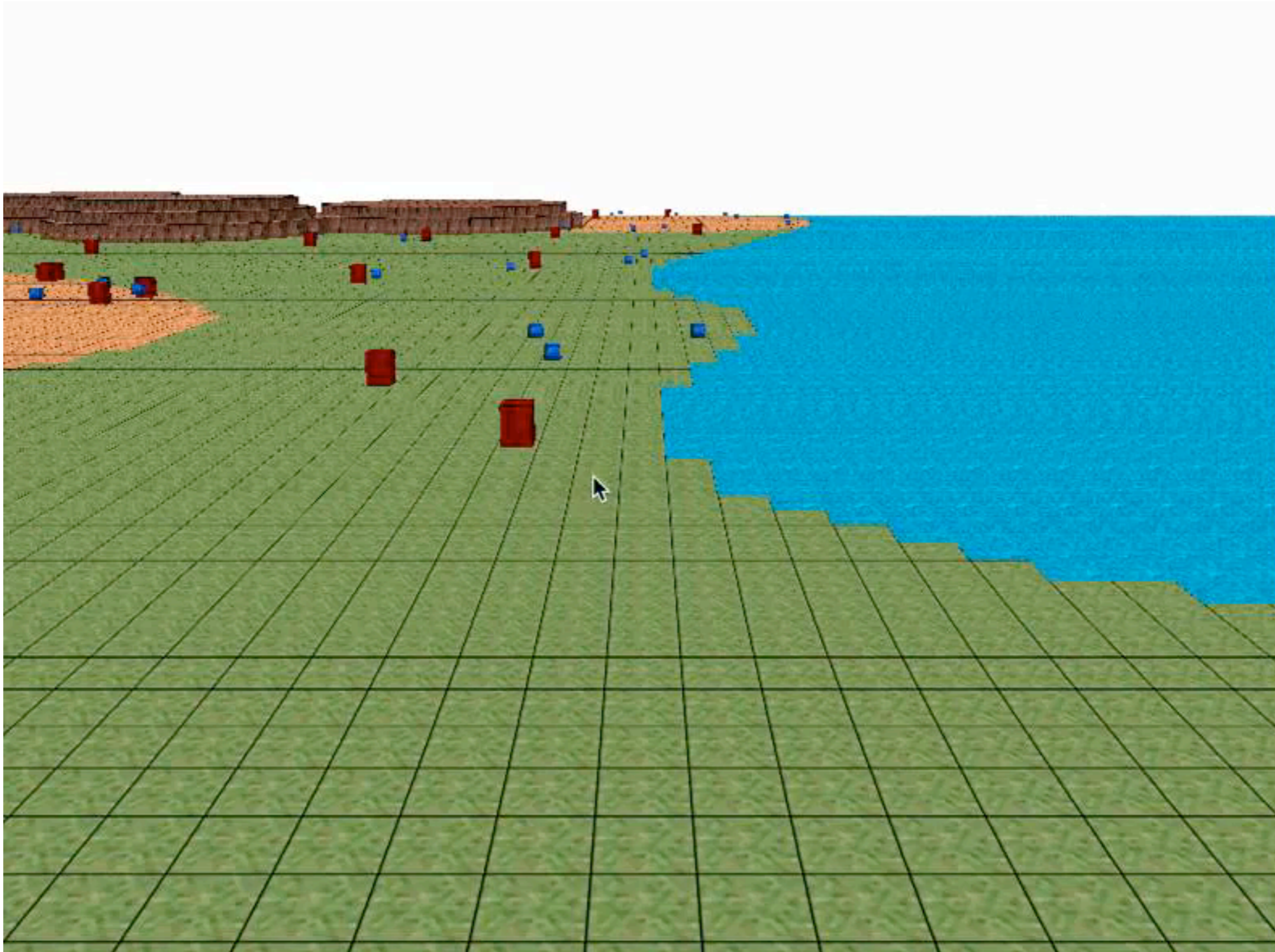
---

Argument

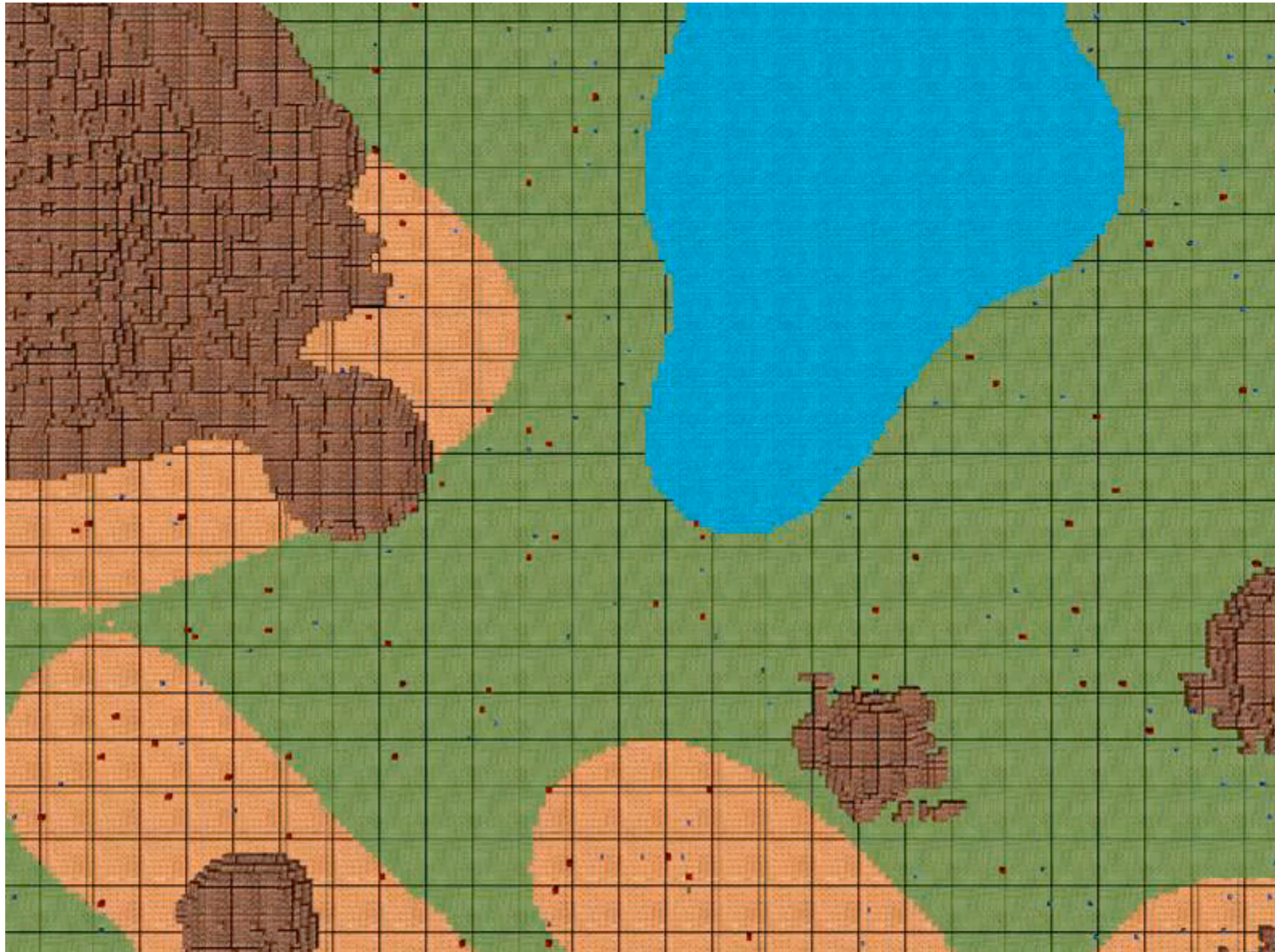
Realization

Experience

# Experiments: spatialDictionary



# Experiments: spatialDictionary



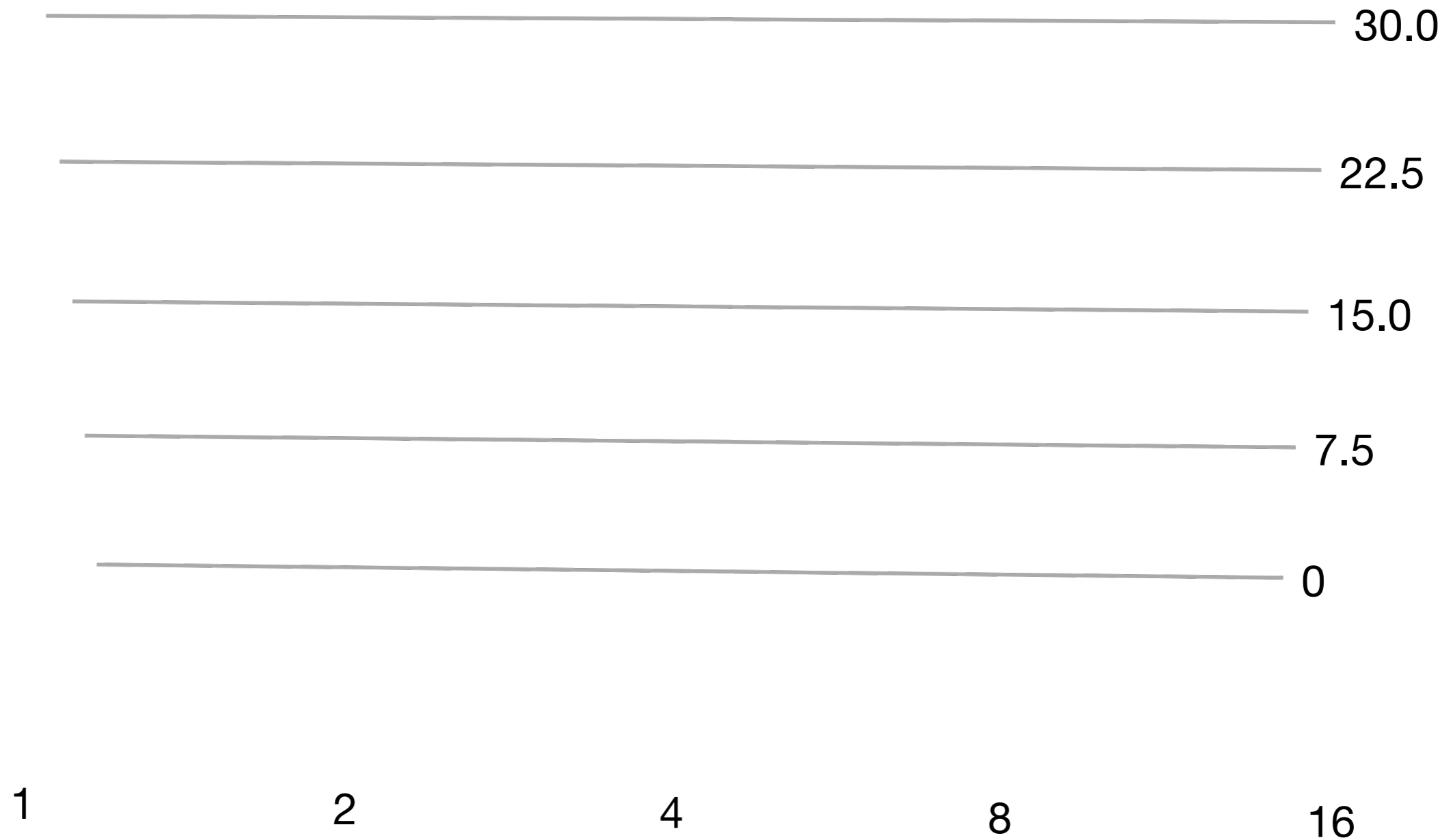


# Experiments: spatialDictionary

---

- Global Lock
- SvS
- Progressive Lock
- SvS Cached

# Experiments: spatialDictionary

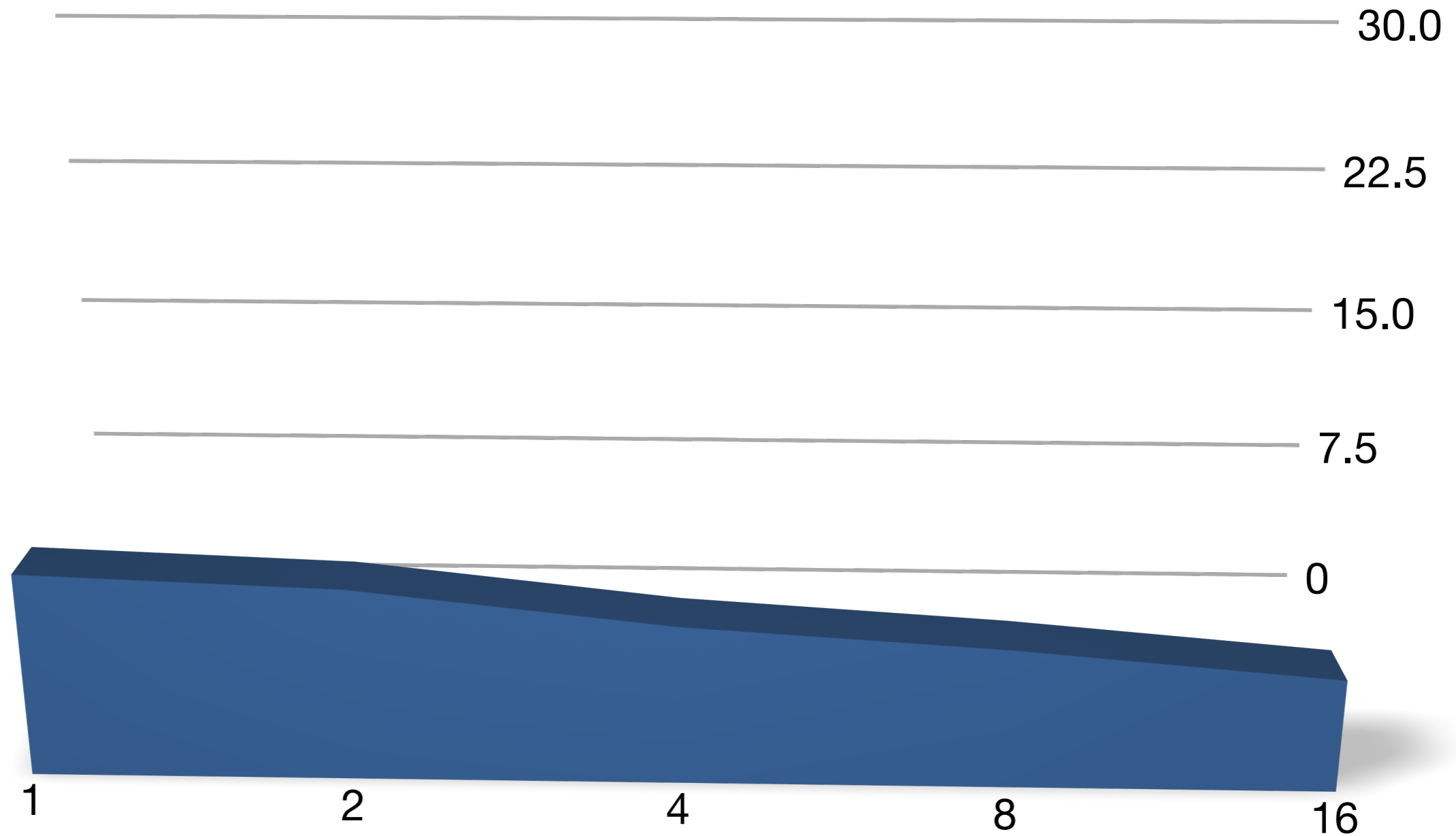


Global Lock  
SvS

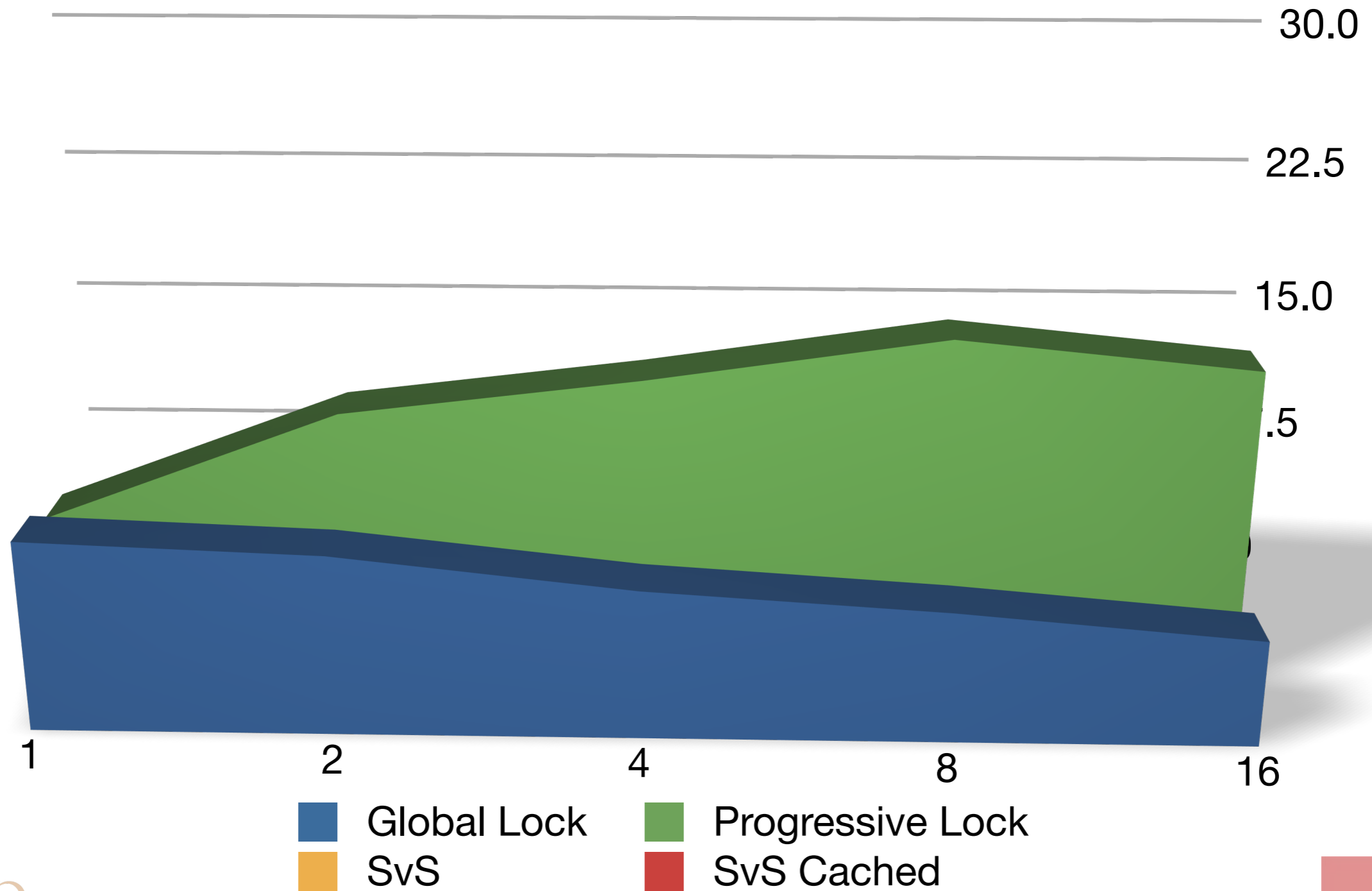
Progressive Lock  
SvS Cached



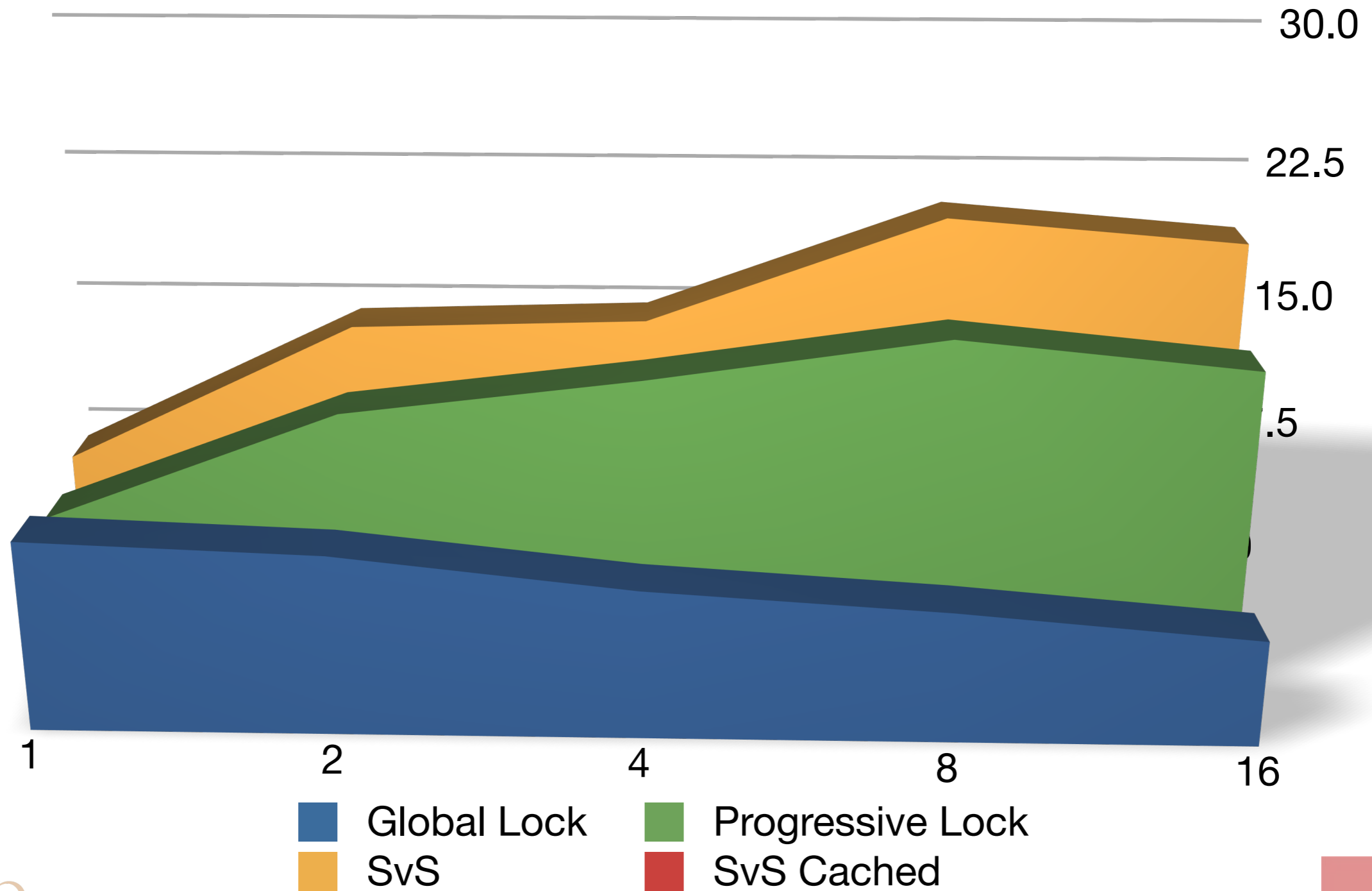
# Experiments: spatialDictionary



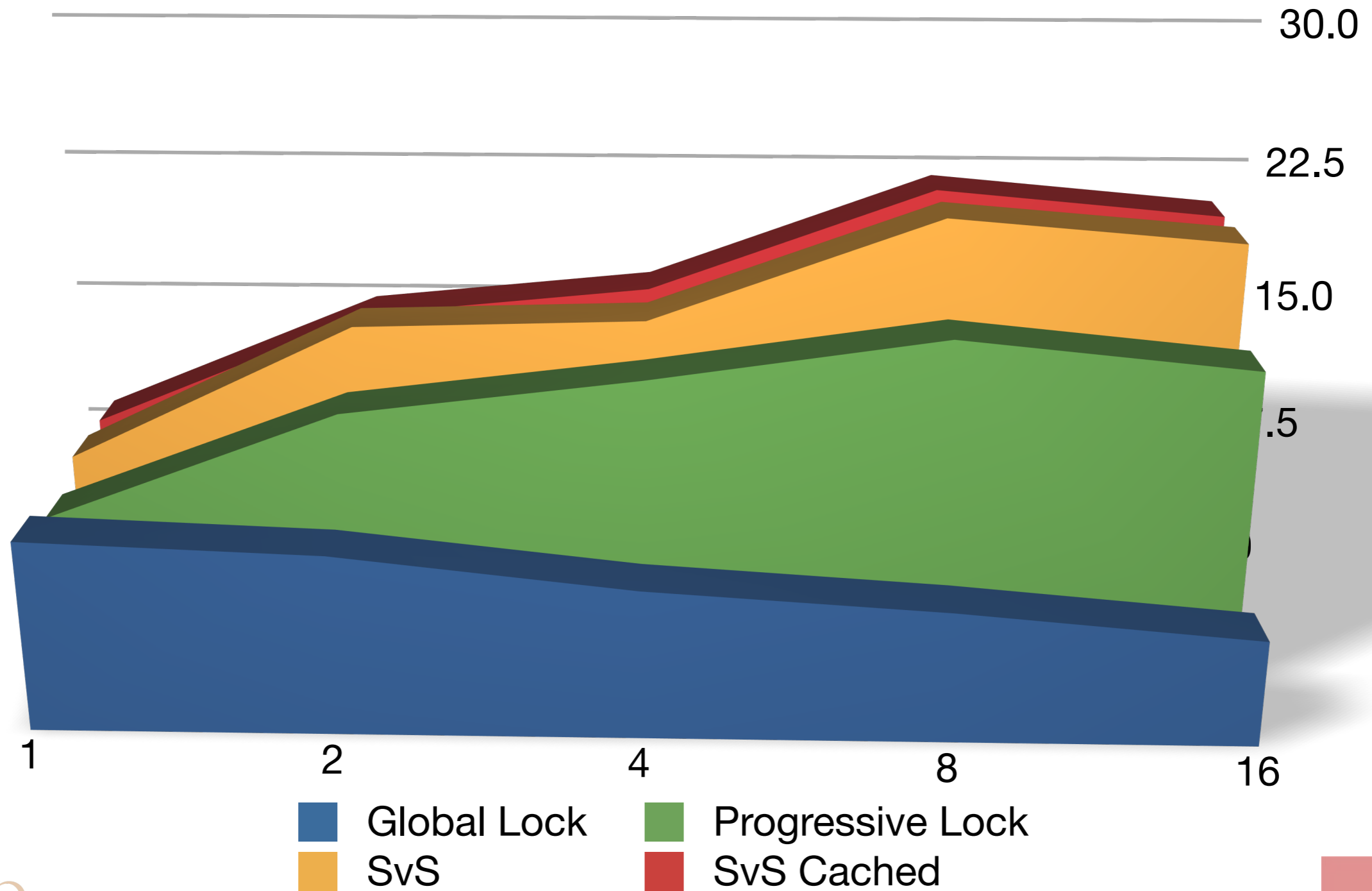
# Experiments: spatialDictionary



# Experiments: spatialDictionary



# Experiments: spatialDictionary





# Future Work

---



# Future Work

---

Optimized scheduling



# Future Work

---

Optimized scheduling

Robust query support

# Questions

---

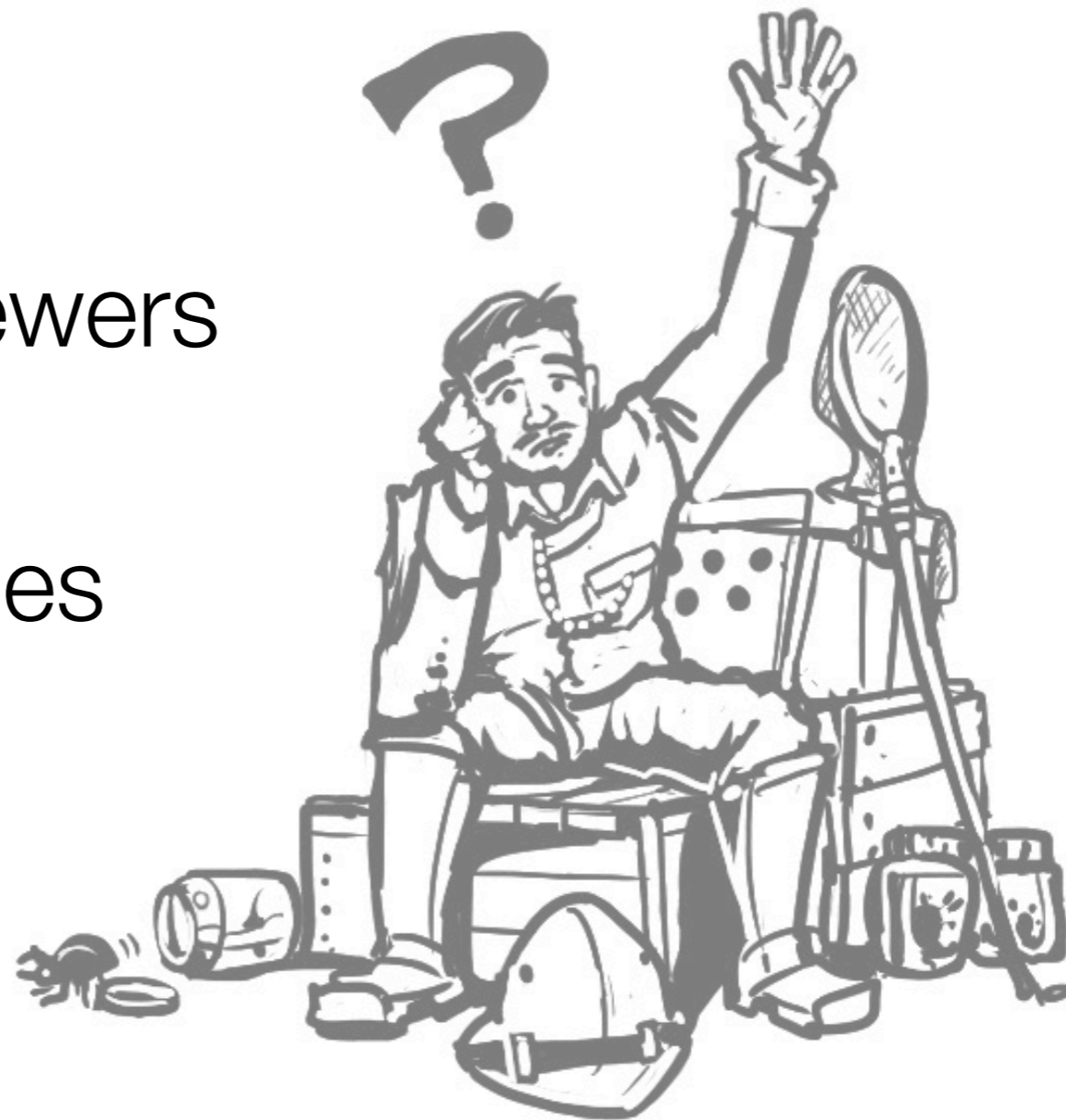




# Questions

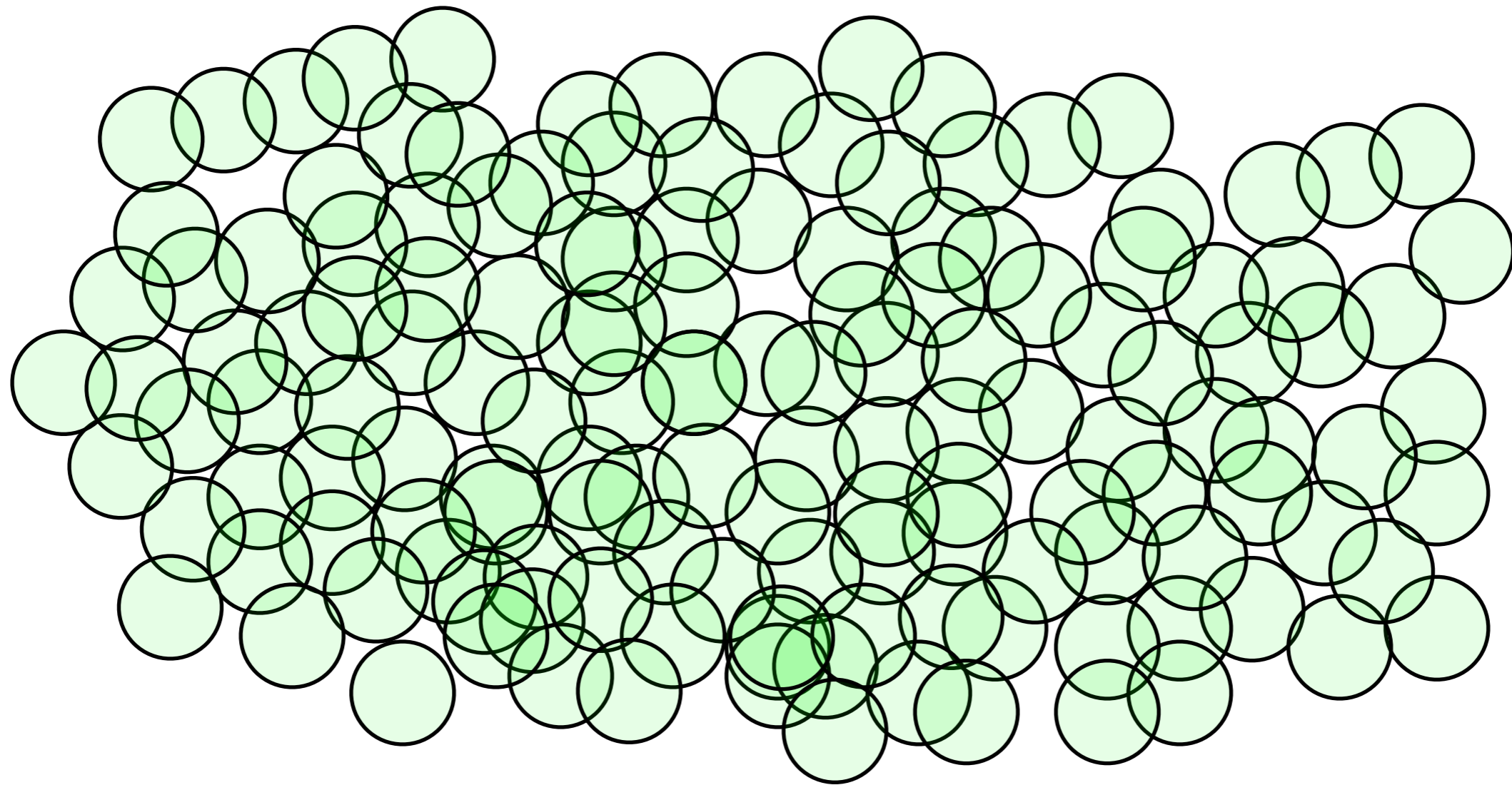
---

Thanks to  
Anonymous reviewers  
Gaslamp Games

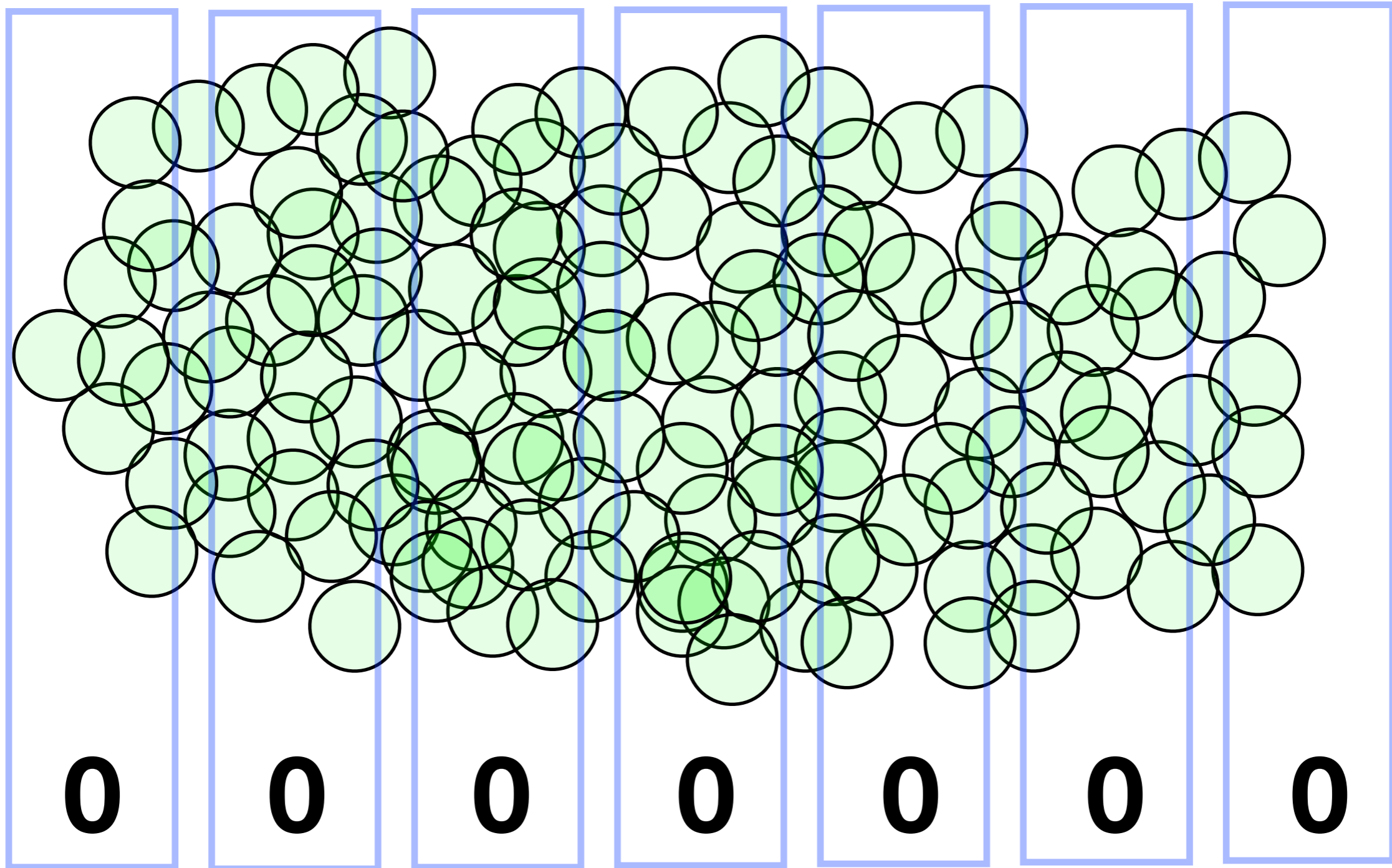


# SvS - Signatures

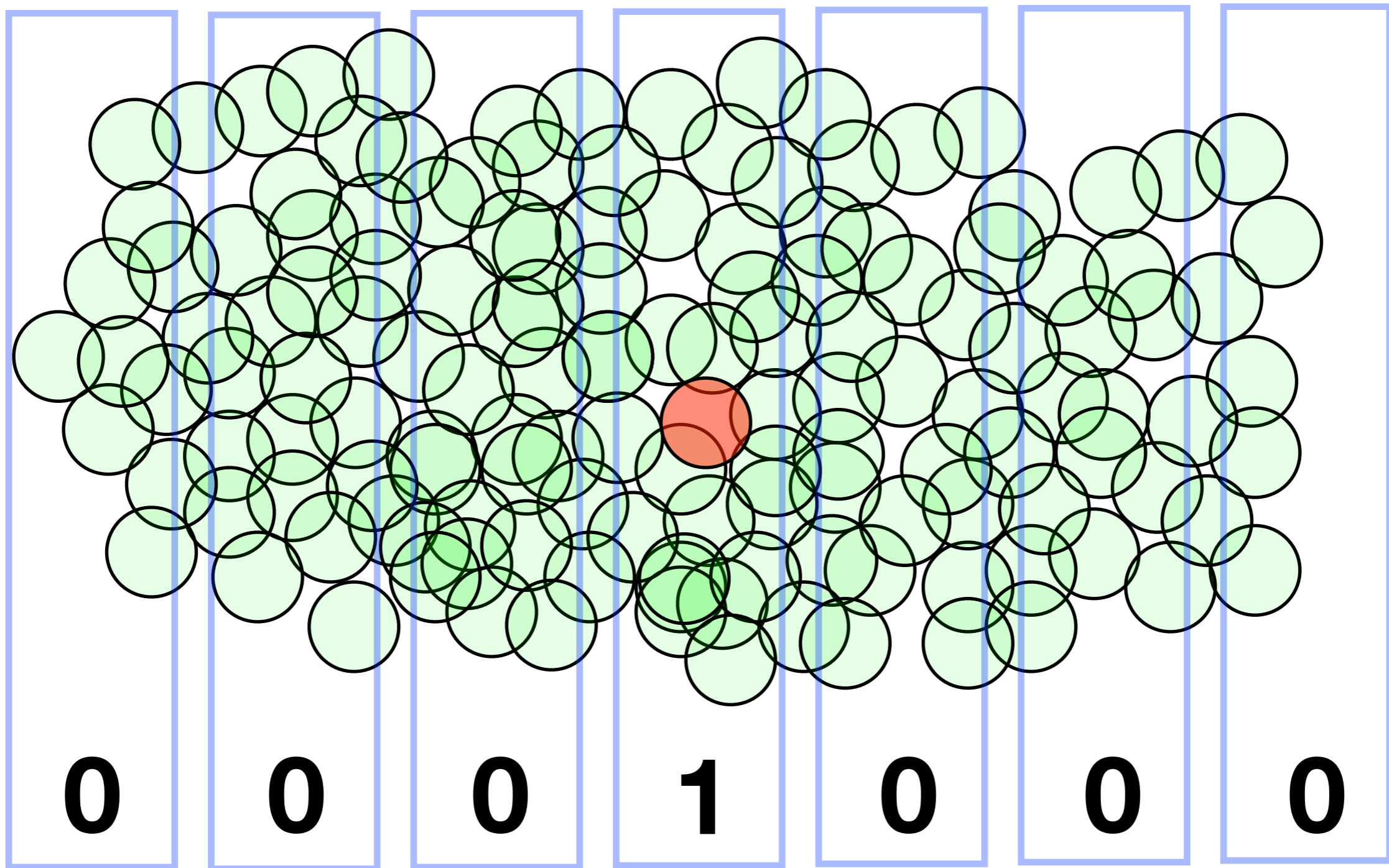
---



# SvS - Signatures



# SvS - Signatures





# SvS - Signatures

---



# SvS - Signatures

---

composable

# SvS - Signatures

---

composable

quick to compute  
and compare

# SvS - Signatures

---

composable

only false positives for  
intersection

quick to compute  
and compare



# SvS - Signatures

---

composable

easy to make arbitrarily  
precise

only false positives for  
intersection

quick to compute  
and compare

# SvS - Signatures

---

composable

easy to make arbitrarily  
precise

only false positives for  
intersection

quick to compute  
and compare

separate evaluation of data  
items from synchronization



# Preliminary Stuff

---



# Preliminary Stuff

---

```
inXspan( int x, int s ) : Cell I [ | I.x - x | <= s ]  
inYspan( int y, int s ) : Cell I [ | I.y - y | <= s ]  
inRadius( ivec2 C, int r ) : Cell I [ inXspan<C> ( C.x, r ) and  
inYspan<C>( C.y, r ) ]
```

↑  
Super preliminary warning!