Performance and Scalability

(Chapter 11)

Performance and Scalability

- Performance: How long is the latency?
- Scalability: Do we get higher throughput if we add more resources?

Performance and Scalability

- Performance: How long is the latency?

versus

- Scalability: Do we get higher throughput if we add more resources?

Performance and Scalability

- Performance: How long is the latency?

versus

- Scalability: Do we get higher throughput if we add more resources?

storage

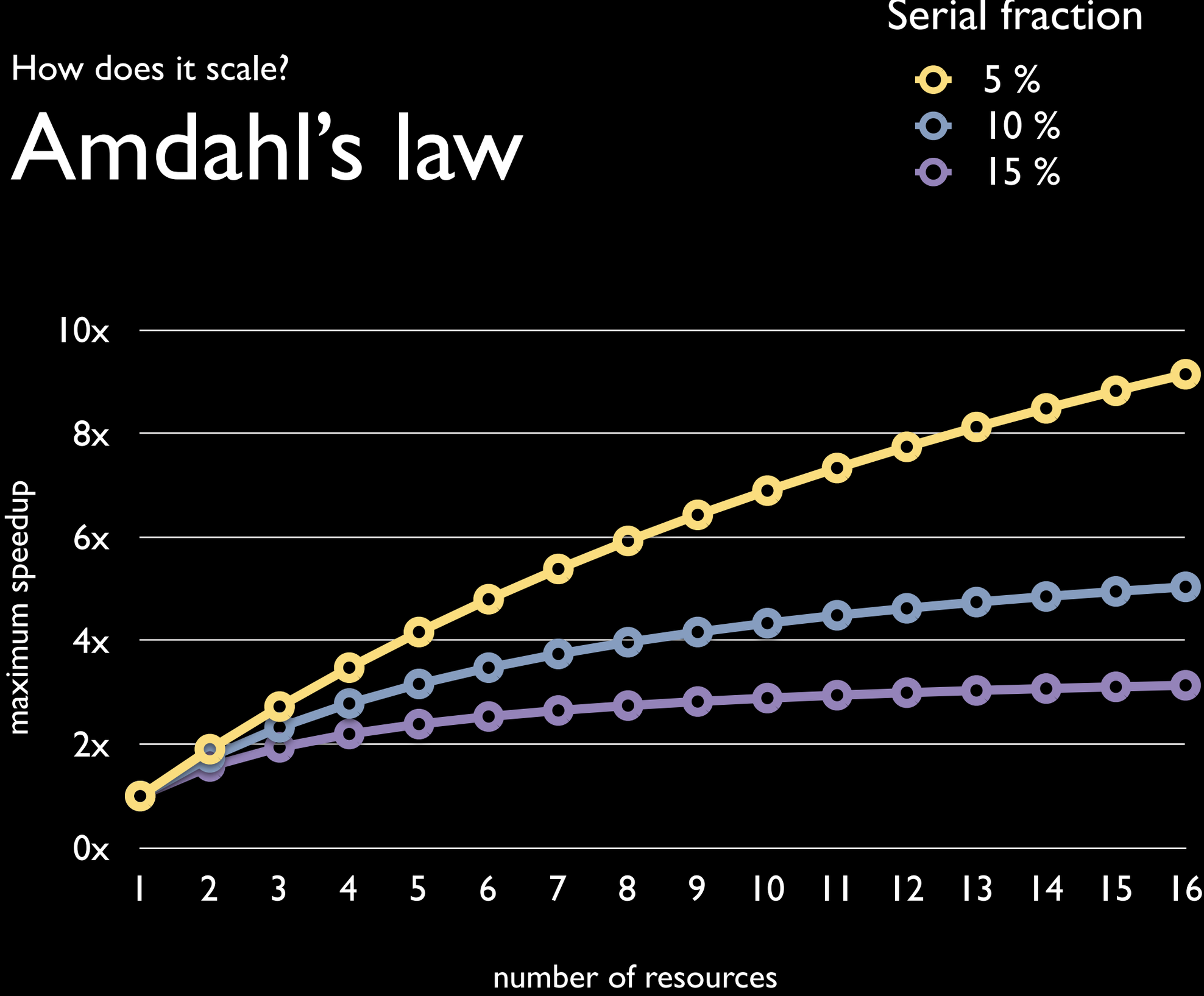
CPUs

I/O bandwidth

memory

How does it scale?

Amdahl's law



- Amdahl's law shows that serial execution harms scalability.
- Locks guarantee serial access.
- Contended locking harms scalability.
- How can we reduce lock contention?

Reducing Lock Contention

- Narrowing lock scope
- Reducing lock granularity
  - Lock splitting
  - Lock striping
- Alternatives to exclusive locks

Reducing Lock Contention

Narrowing lock scope

```
@ThreadSafe
public class AttributeStore {
    @GuardedBy("this")
    private final Map<String, String>
        attributes = new HashMap<String, String>();

    public synchronized boolean userLocationMatches(
            String name, String regexp) {

        String key = "users." + name + ".location";

        String location = attributes.get(key);

        if (location == null)
            return false;
        else
            return Pattern.matches(regexp, location);
    }
}
```

Reducing Lock Contention

Narrowing lock scope

```
@ThreadSafe
public class AttributeStore {
    @GuardedBy("this")
    private final Map<String, String>
        attributes = new HashMap<String, String>();

    public boolean userLocationMatches(
            String name, String regexp) {

        String key = "users." + name + ".location";
        synchronized (this) {
            String location = attributes.get(key);
        }
        if (location == null)
            return false;
        else
            return Pattern.matches(regexp, location);
    }
}
```

Reducing Lock Contention

Reducing lock granularity

```
@ThreadSafe
public class ServerStatus {
    @GuardedBy("this") public final Set<String> users;
    @GuardedBy("this") public final Set<String> queries;
    ...
    public synchronized void addUser(String u) {
        users.add(u);
    }
    public synchronized void addQuery(String q) {
        queries.add(q);
    }
    public synchronized void removeUser(String u) {
        users.remove(u);
    }
    public synchronized void removeQuery(String q) {
        queries.remove(q);
    }
}
```

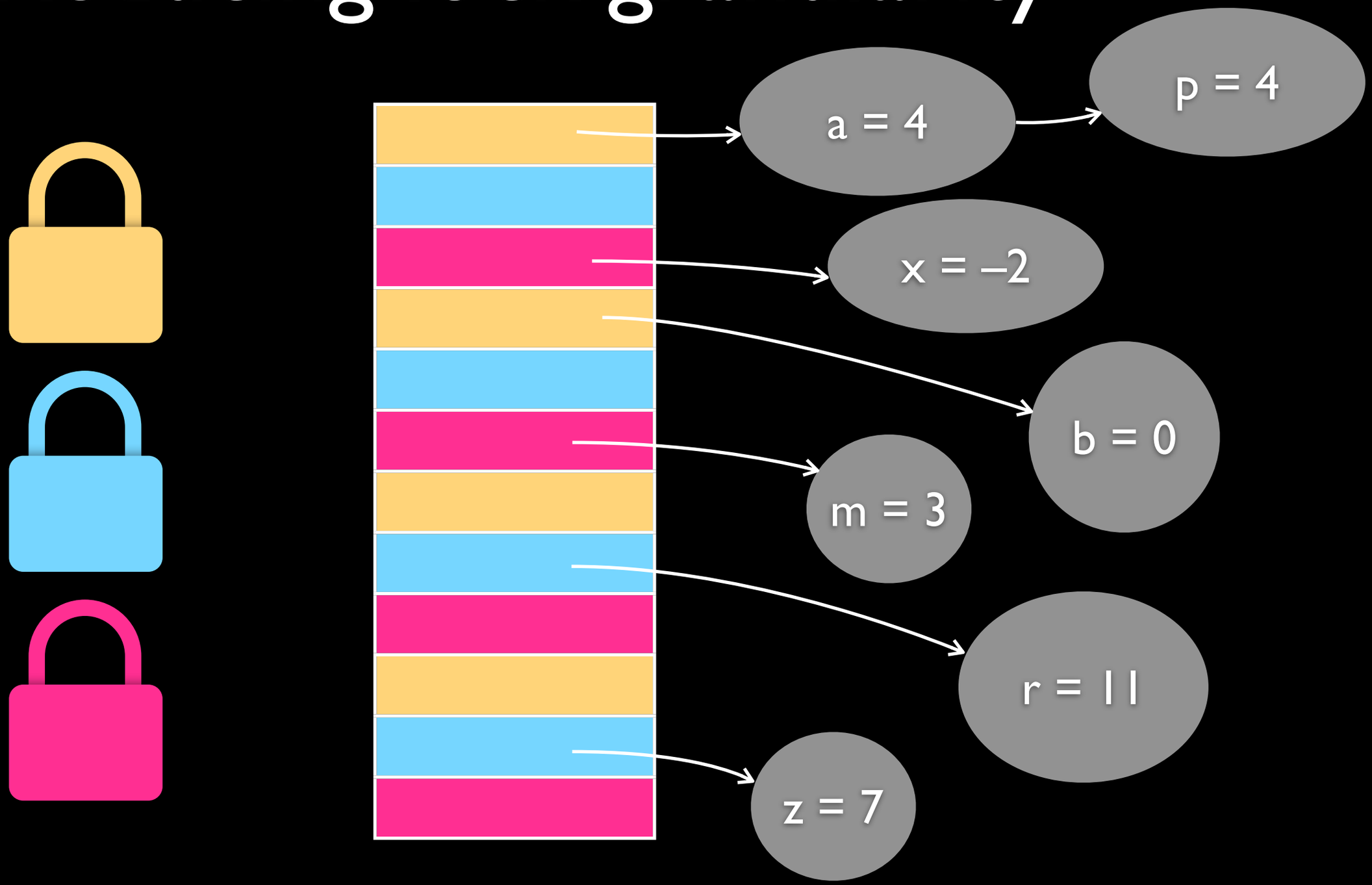
Reducing Lock Contention

Reducing lock granularity

```
@ThreadSafe
public class ServerStatus {
    @GuardedBy("users") public final Set<String> users;
    @GuardedBy("queries") public final Set<String> queries;
    ...
    public void addUser(String u) {
        synchronized (users) {
            users.add(u);
        }
    }
    public void addQuery(String q) {
        synchronized (queries) {
            queries.add(q);
        }
    }
    // removeUser and removeQuery similarly refactored
}
```

Reducing Lock Contention

Reducing lock granularity



Alternatives to exclusive locks

- Immutable objects (ch. 3)
- Concurrent collections (ch. 5)
- Read-write locks (ch. 13)
- Atomic variables (ch. 15)

When should we do this?

NEVER!!!

(Unless your program does not fulfill the performance requirements)

If the program does not fulfill the performance requirements…

Where should we start?

MEASURE!!!

(To find the bottle neck)

Thank you

Exercise

Coming soon…