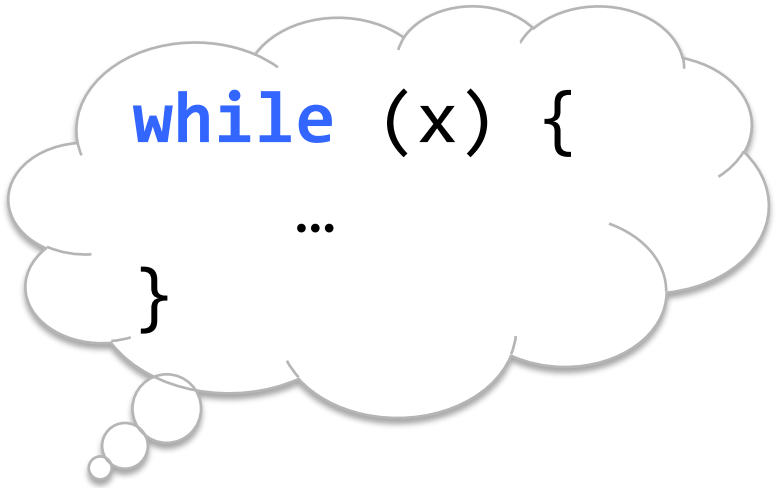
Non-deterministic parallelism considered useful

Derek G. Murray Steven Hand

University of Cambridge

1. **Parallelization**
2. **Synchronization**
3. **Scheduling**
4. **Load balancing**
5. **Communication**
6. **Fault tolerance**
7. **Guaranteed termination**

1. Parallelization
2. Synchronization
3. Scheduling
4. Load balancing
5. Communication
6. Fault tolerance
7. ~~Guaranteed termination~~



1. Parallelization
2. Synchronization
3. Scheduling
4. Load balancing
5. Communication
6. Fault tolerance ⇔ Deterministic
7. ~~Guaranteed termination~~

Real programmers don't use

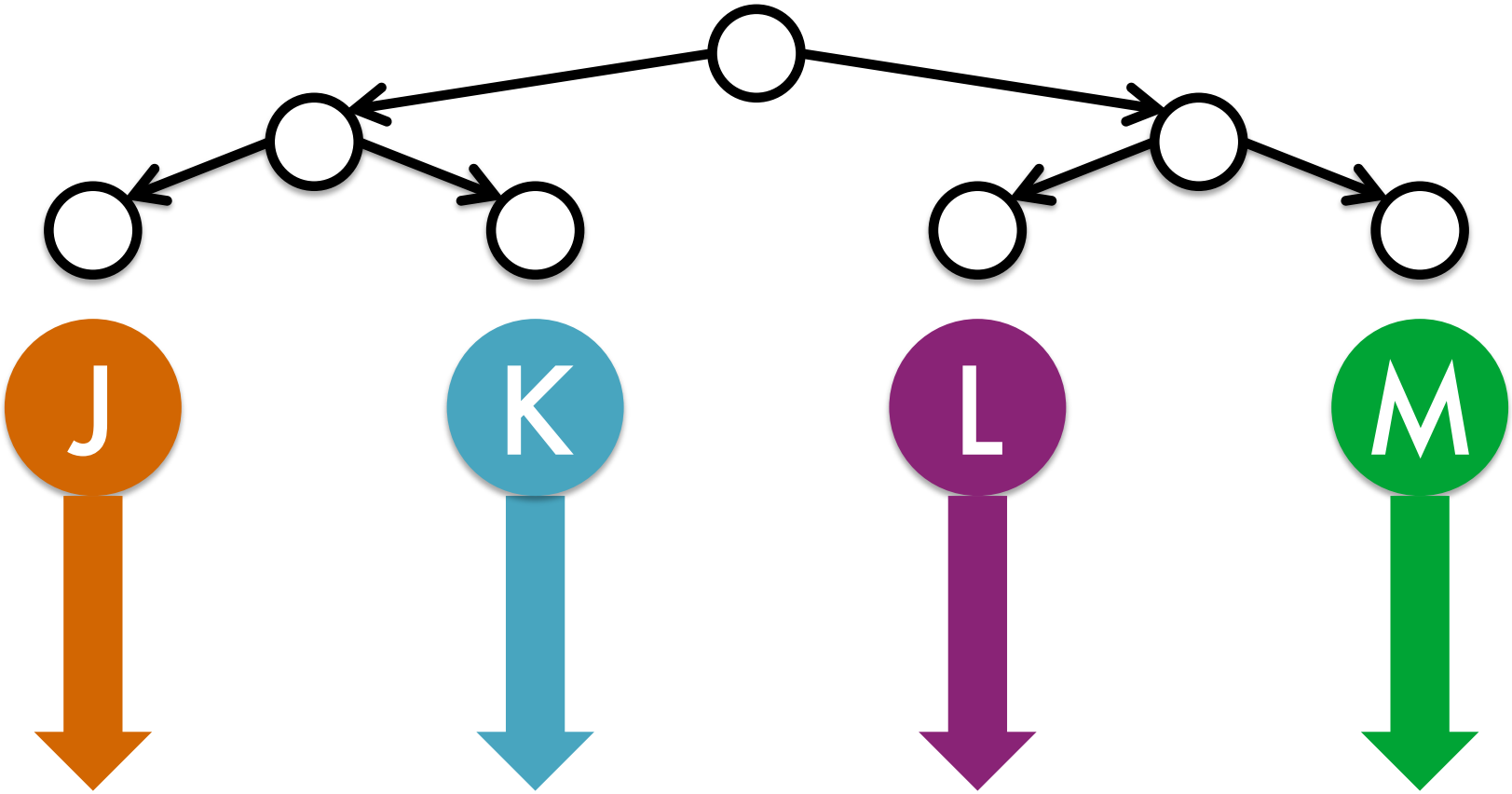
deterministic parallelism

Real programmers use

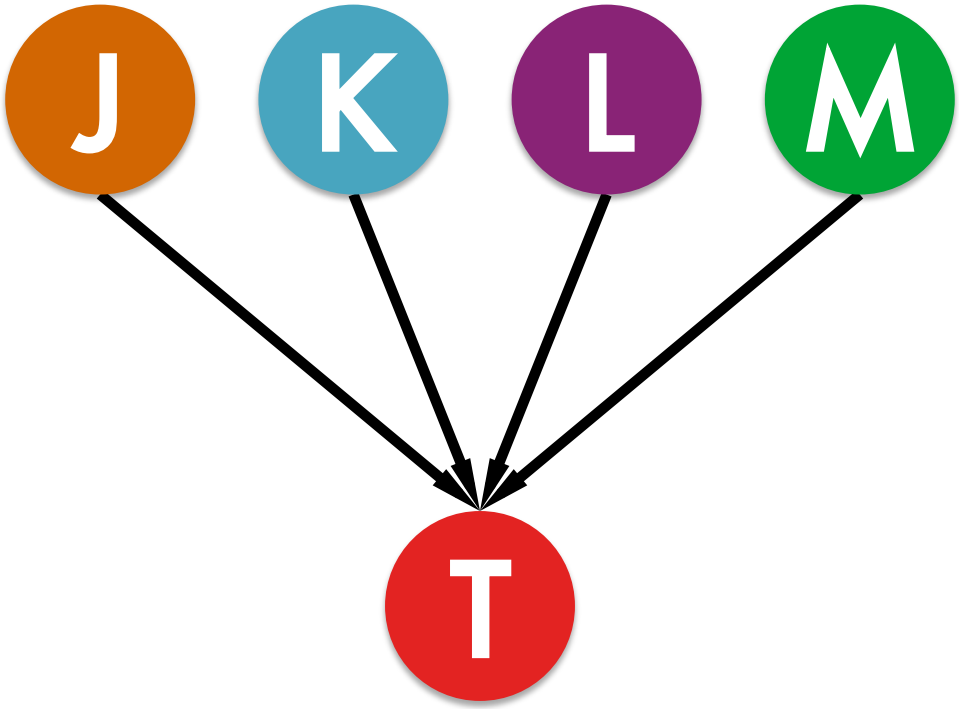
- Async. Networks
- Real hardware
- Performance interference
- User input
- Timeouts
- Signals
- `select()` loops
- Condition variables
- Mutable state

All of these cause non-determinism!

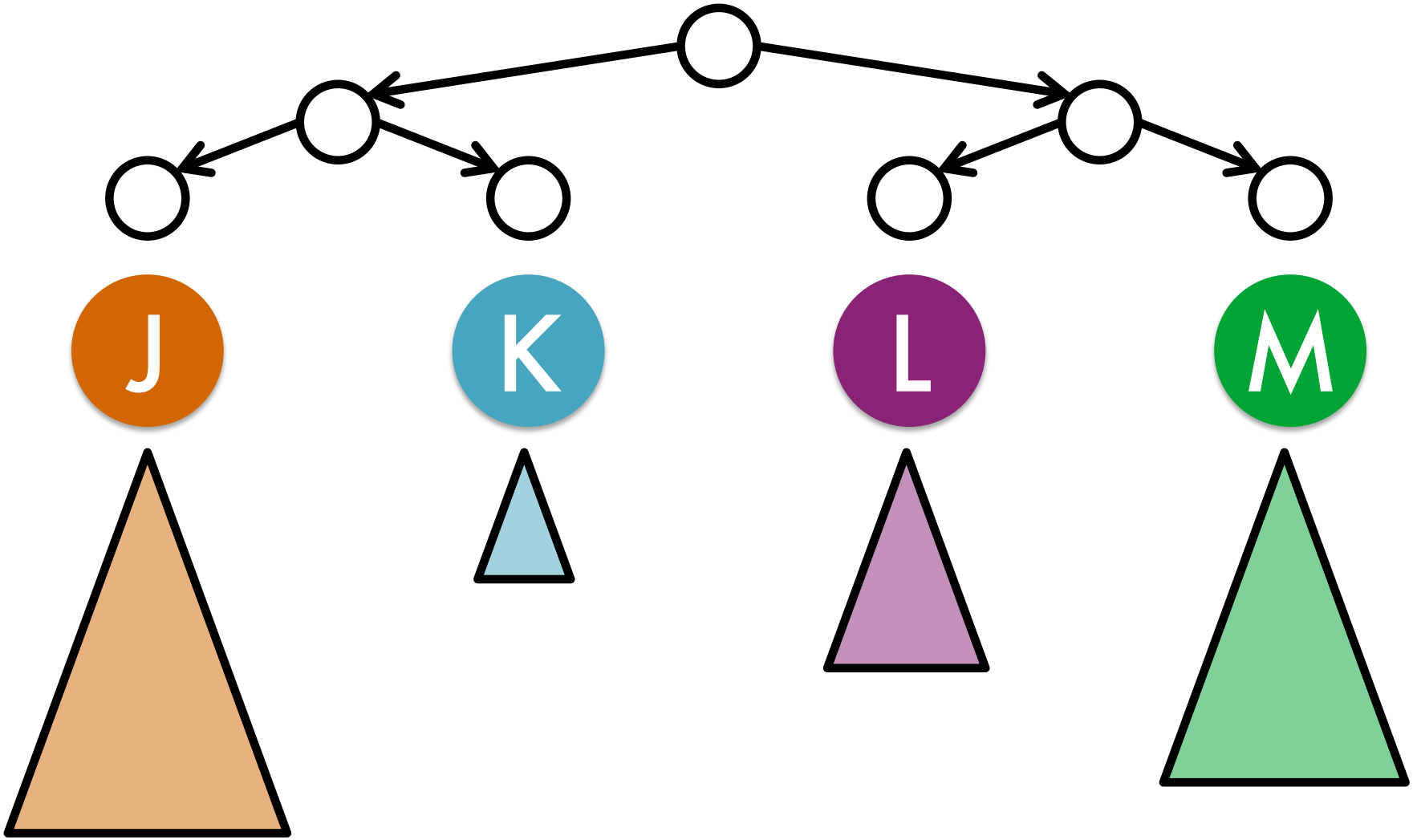
Example: branch-and-bound



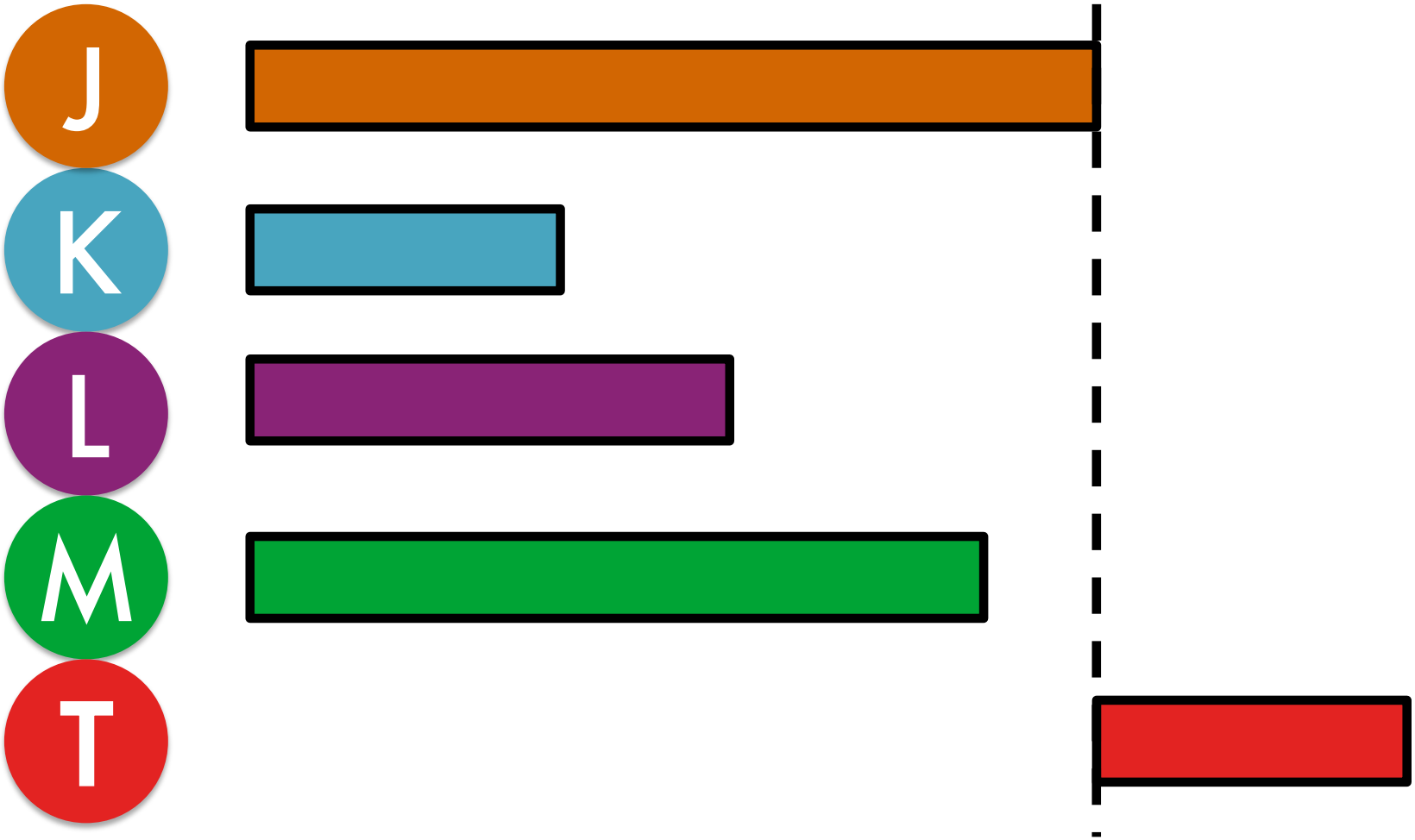
Deterministic data flow



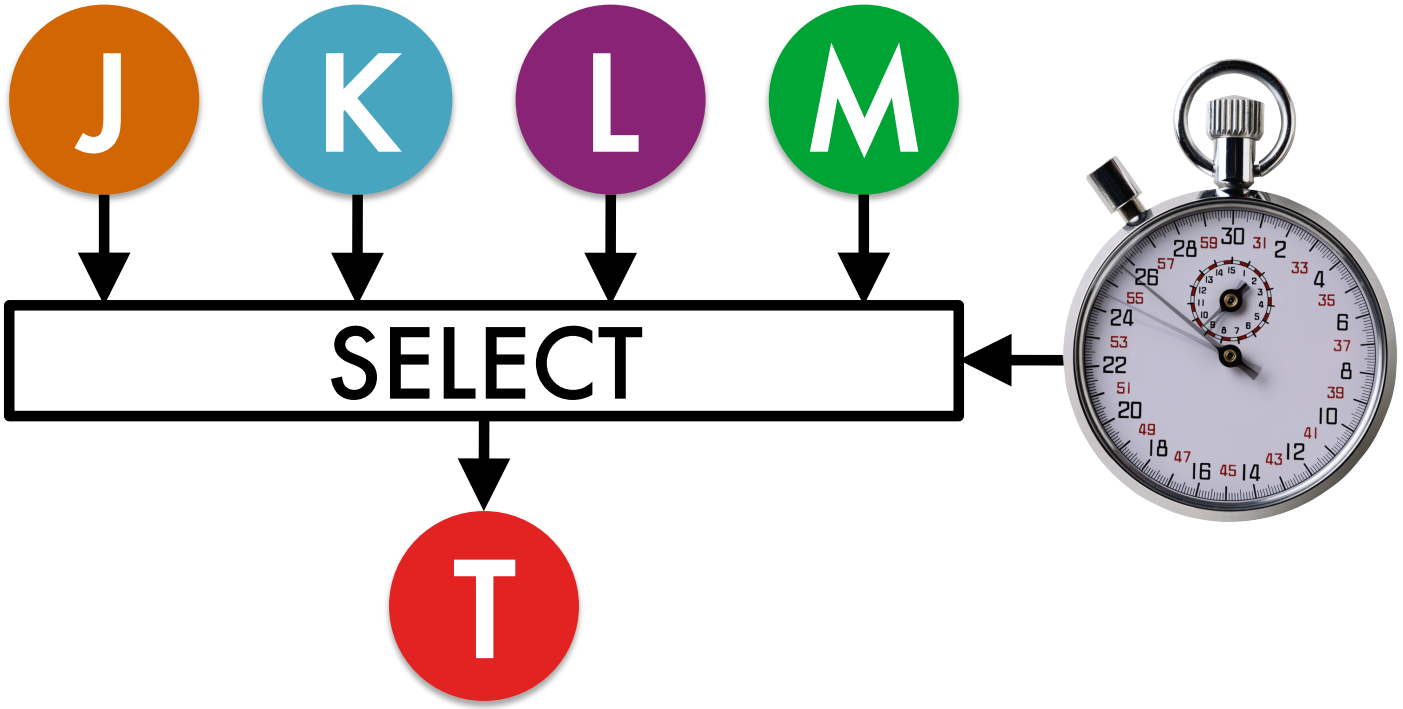
Irregular parallelism



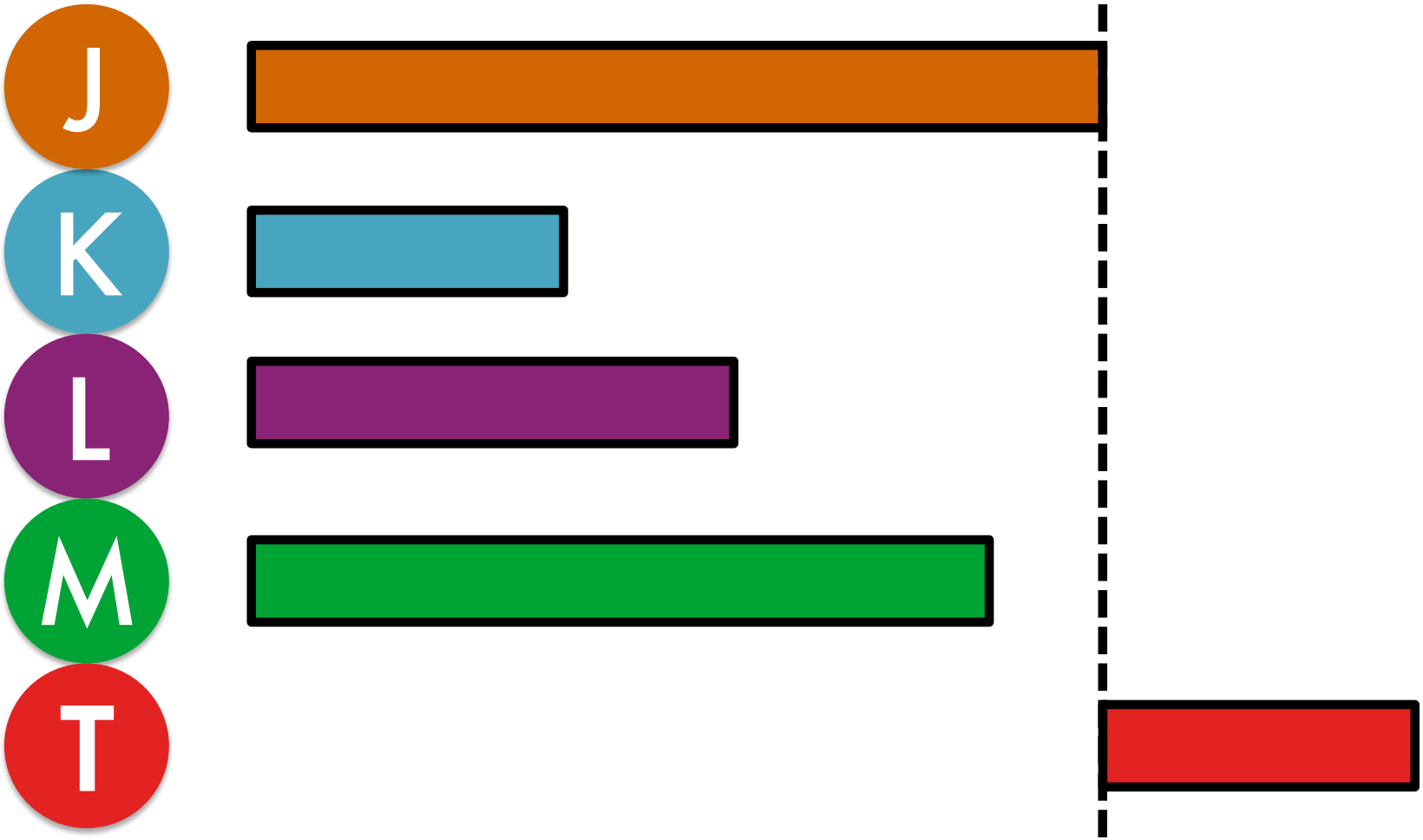
Irregular parallelism



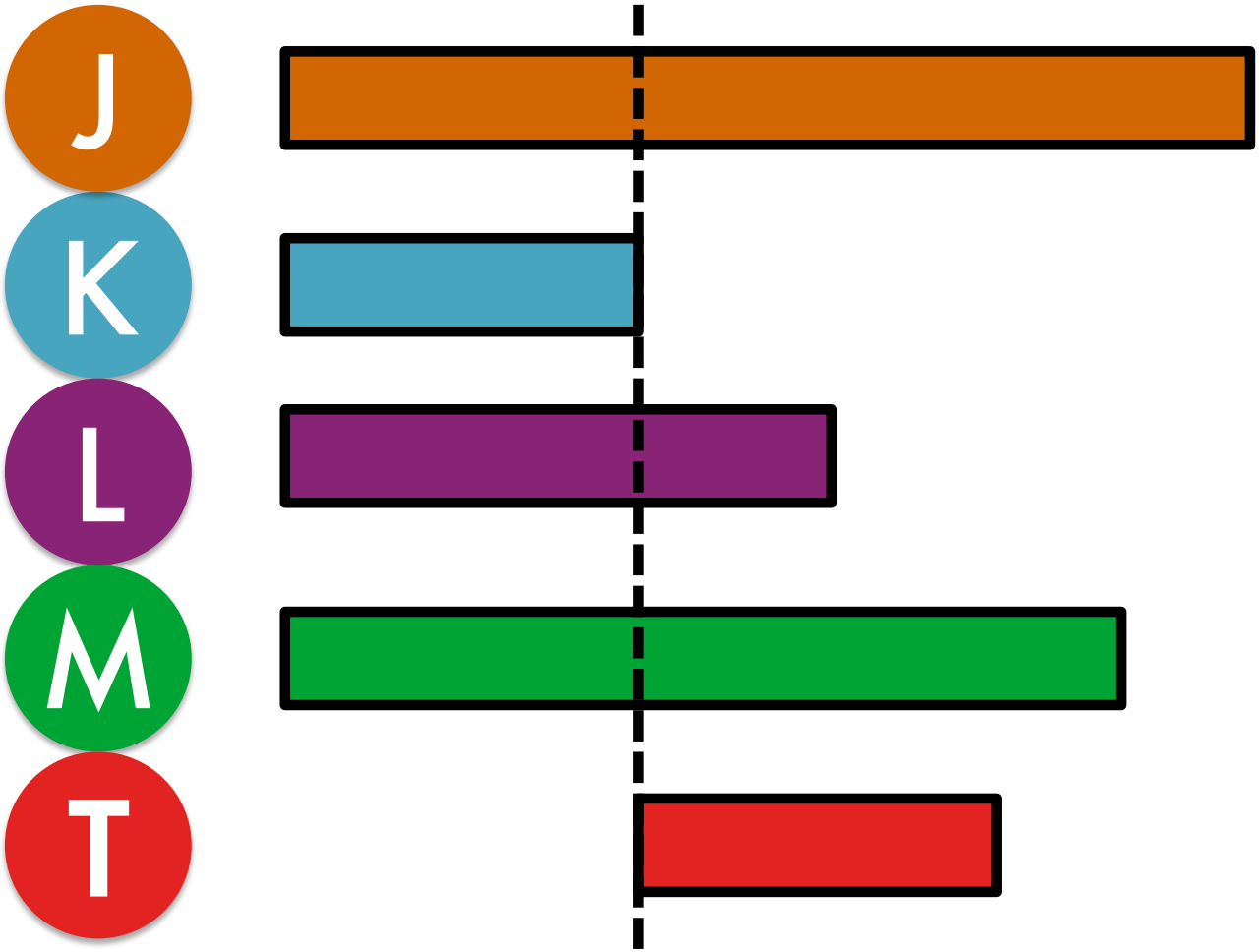
Non-deterministic select



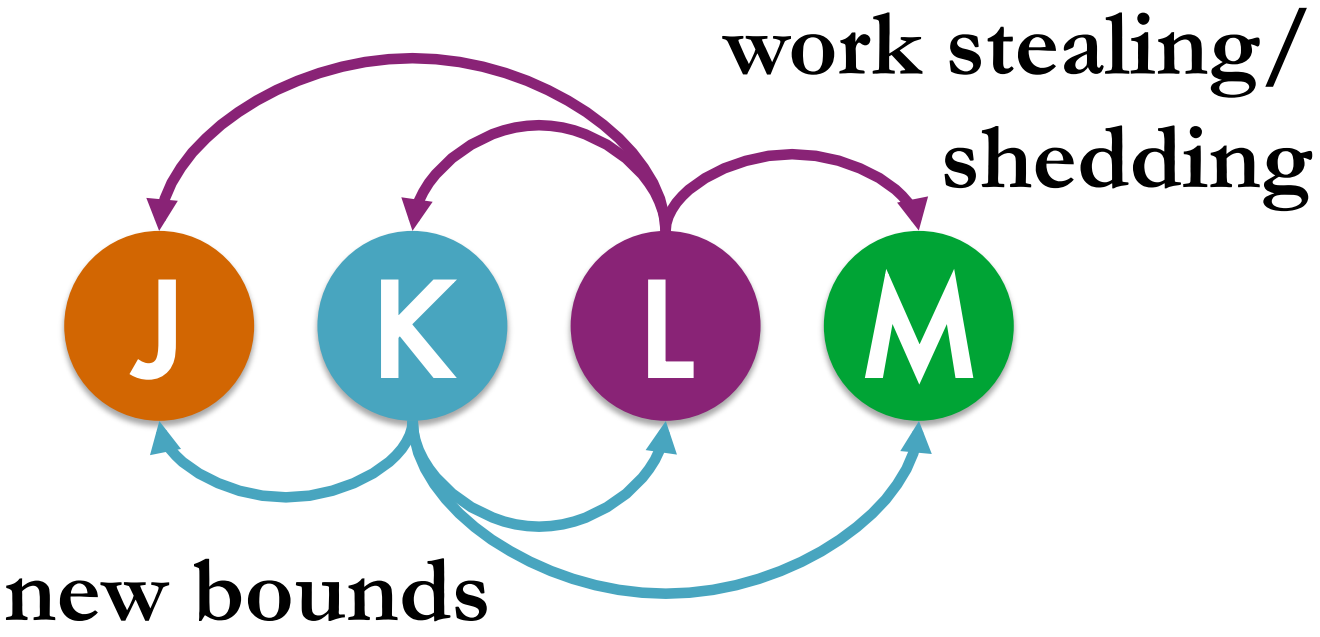
Non-deterministic select



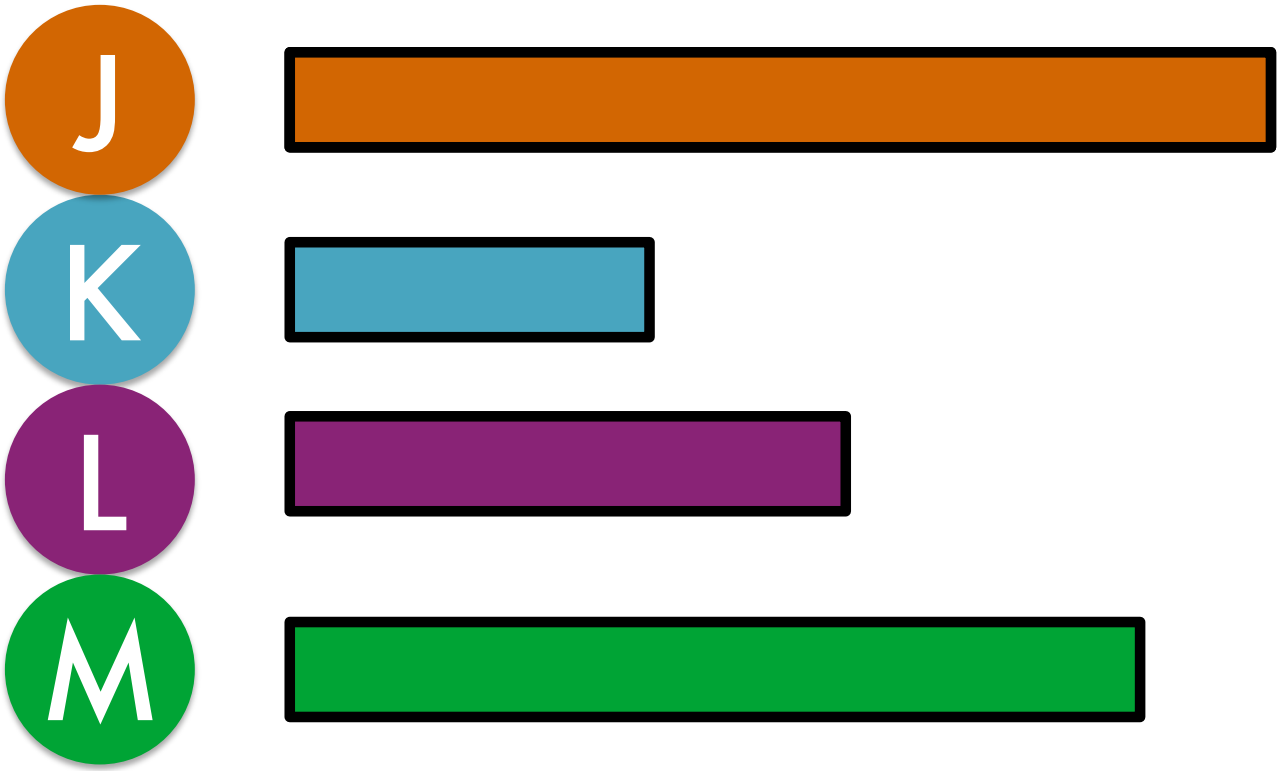
Non-deterministic select



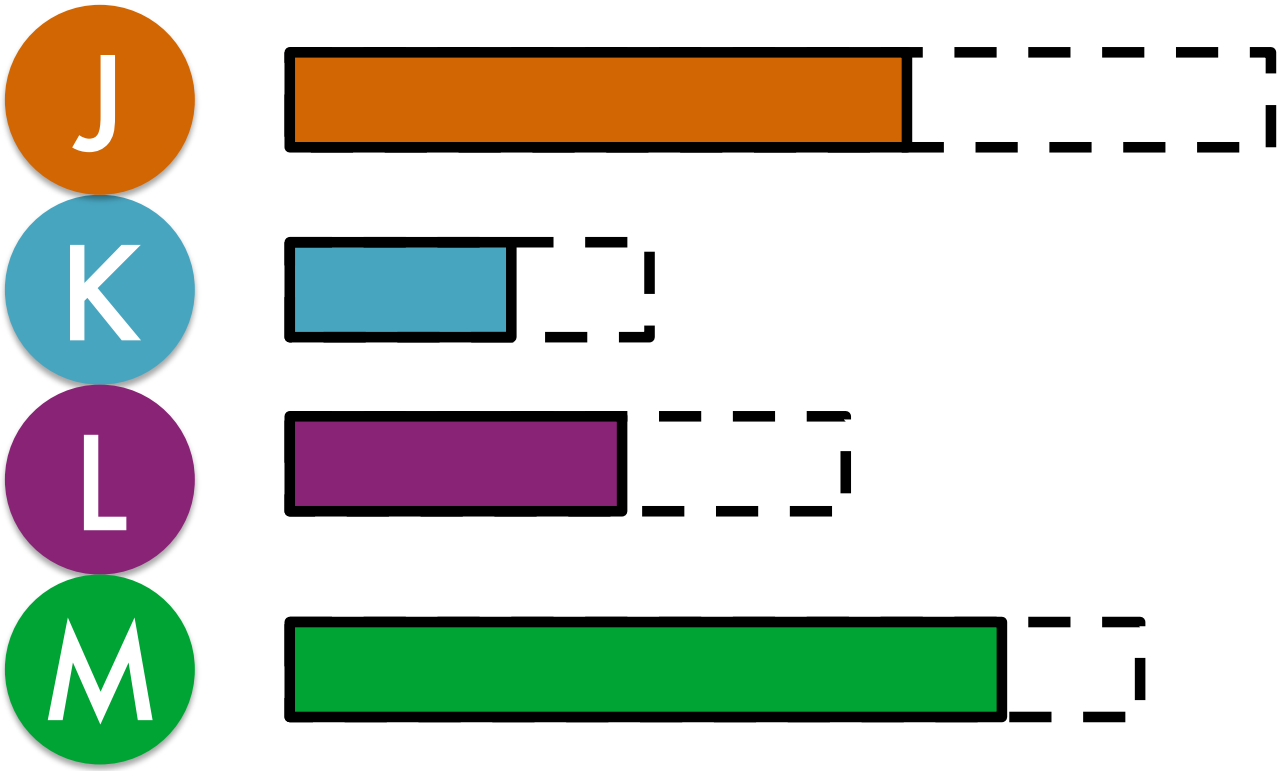
Asynchronous signals



Asynchronous signals



Asynchronous signals



1. Parallelization
2. Synchronization
3. Scheduling
4. Load balancing
5. Communication
6. Fault tolerance
7. ~~Guaranteed termination~~

1. Parallelization
2. Synchronization
3. Scheduling
4. Load balancing
5. Communication
6. ~~Fault tolerance~~
7. ~~Guaranteed termination~~

Challenge: dealing with faults

- Fail everything
  - Error codes/exceptions
  - Bounded non-determinism
  - Checkpoints
- Record and replay

Conclusions

- Many benefits of non-determinism
  - Performance, adaptability, interactivity
- System must allow non-determinism
- Determinism at language-level
  - For programmers who need training wheels