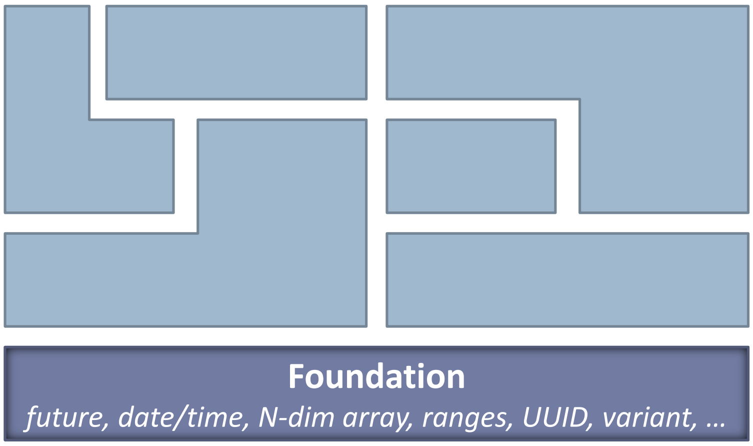
## Library Composability

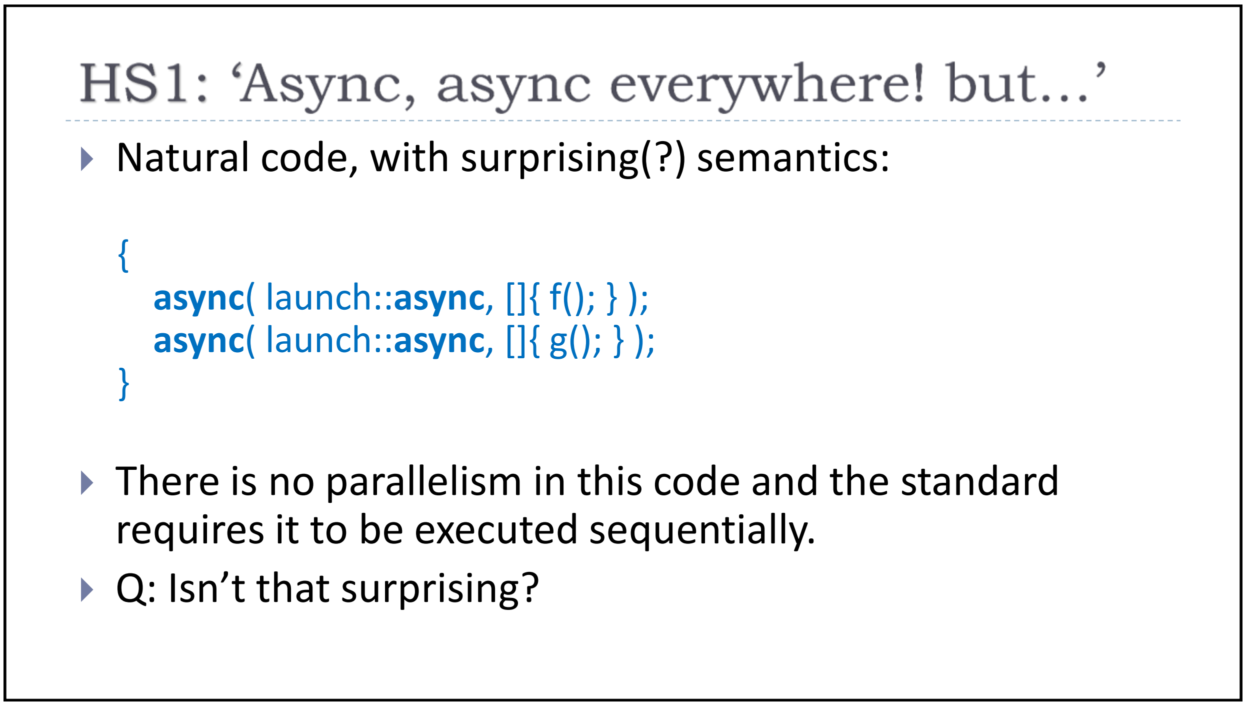


## HB1: Lifetime error (N2802)

- "// Don't write this code!!!"

```
int fib(int n) {
   if (n <= 1) return n;
   int fib1, fib2;
   std::thread t( [=, &fib1] { fib1 = fib(n-1); } );
   fib2 = fib(n-2);
   if (fib2 < 0) throw "ick";
   t.join();
   return fib1 + fib2;
}
```

- Lifetime error.

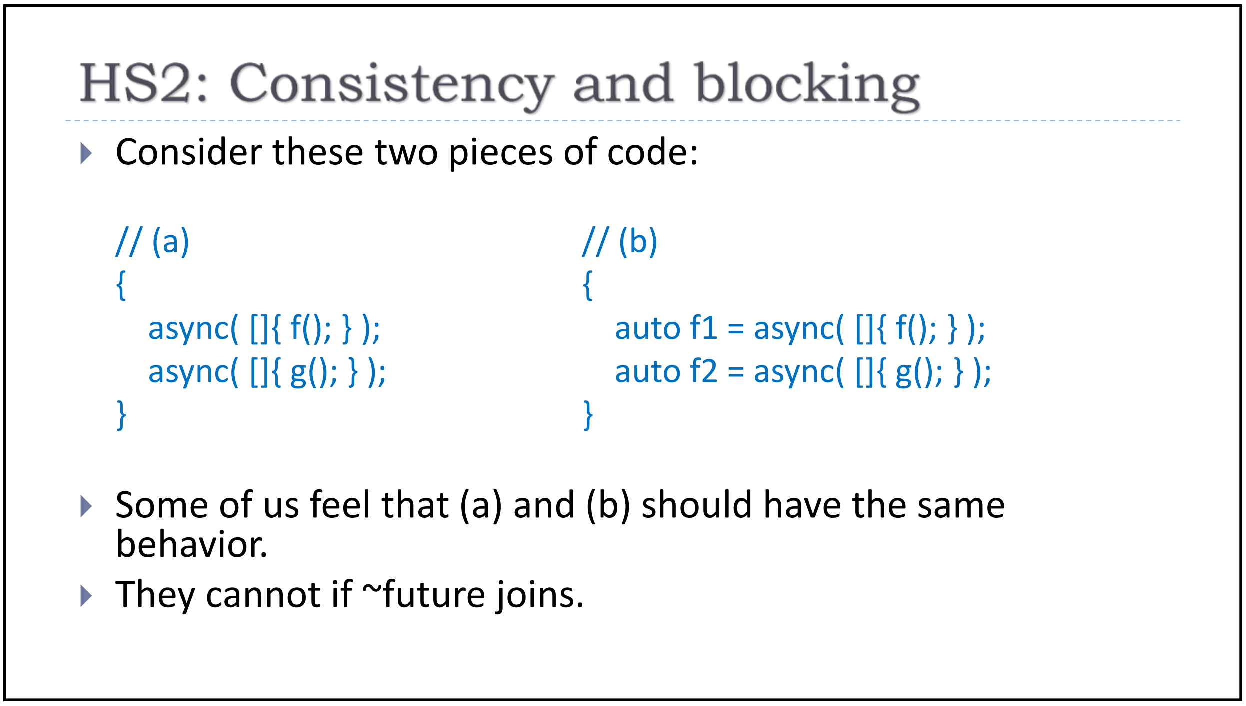


## HS1: 'Async, async everywhere! but…'

- Natural code, with surprising(?) semantics:

```
{
   async( launch::async, []{ f(); } );
   async( launch::async, []{ g(); } );
}
```

- There is no parallelism in this code and the standard requires it to be executed sequentially.
- Q: Isn't that surprising?



## HS2: Consistency and blocking

- Consider these two pieces of code:

```
// (a)                               // (b)
{                                    {
   async( []{ f(); } );                 auto f1 = async( []{ f(); } );
   async( []{ g(); } );                 auto f2 = async( []{ g(); } );
}                                    }
```

- Some of us feel that (a) and (b) should have the same behavior.
- They cannot if ~future joins.

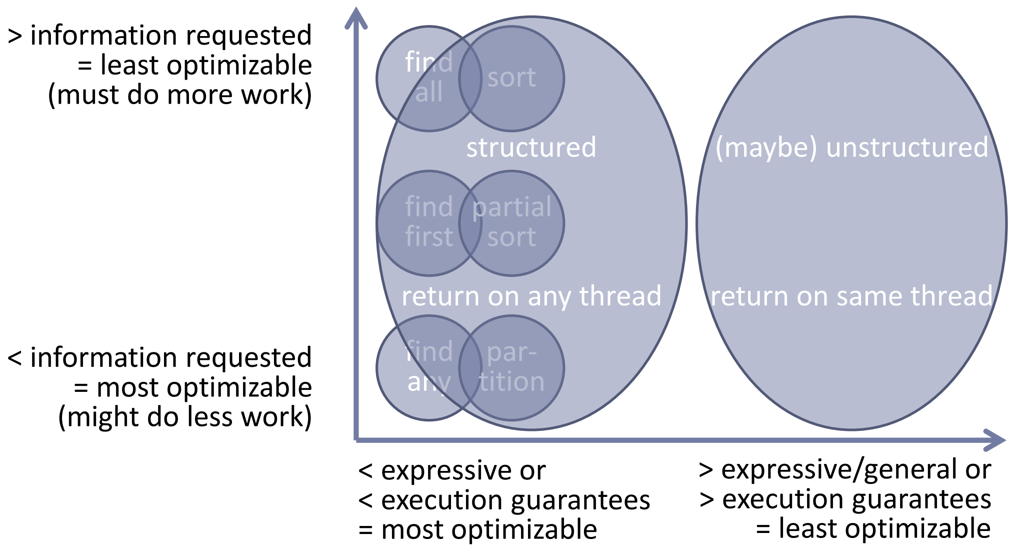
## HS3: Predictable blocking

- What does this code do? In particular, does it block?

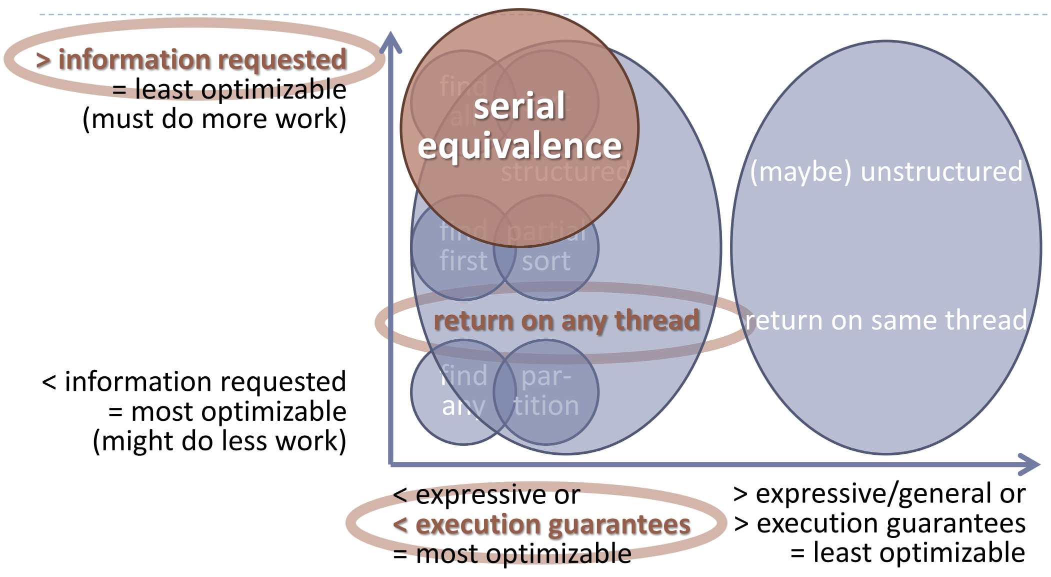
```
void func() {
    future<int> f = start_some_work();
    /*... more code that doesn’t f.get() or f.wait() */
  }
```

  - The answer is different depending on whether the function chose to launch its work via a std::thread or std::async.
- **This is not composable.** We must always be able to tell if code might block.
- **Qs: Can I use std::future**, even though I don’t need the result, if the caller:
  - Could be called under a lock the task may need? **(Deadlock.)**
  - Is supposed to be responsive (e.g., a GUI thread)? **(Nonresponsive.)**

## Two Axes

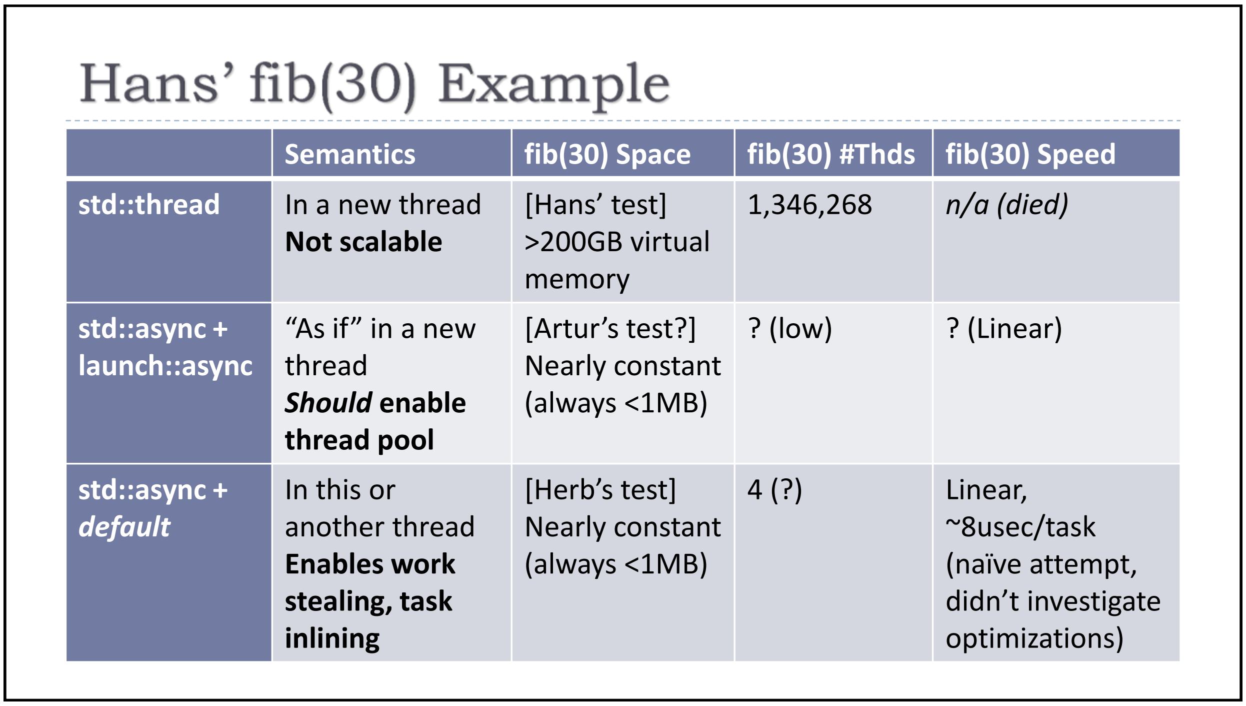


## Two Axes



## Hans' fib(30) Example

| | Semantics | fib(30) Space | fib(30) #Thds | fib(30) Speed |
|---|---|---|---|---|
| **std::thread** | In a new thread **Not scalable** | [Hans' test] >200GB virtual memory | 1,346,268 | *n/a (died)* |
| **std::async + launch::async** | | | | |
| **std::async + *default*** | In this or another thread **Enables work stealing, task inlining** | [Herb's test] Nearly constant (always <1MB) | 4 (?) | Linear, ~8usec/task (naïve attempt, didn't investigate optimizations) |



## Hans' fib(30) Example

| | Semantics | fib(30) Space | fib(30) #Thds | fib(30) Speed |
|---|---|---|---|---|
| **std::thread** | In a new thread **Not scalable** | [Hans' test] >200GB virtual memory | 1,346,268 | *n/a (died)* |
| **std::async + launch::async** | "As if" in a new thread ***Should* enable thread pool** | [Artur's test?] Nearly constant (always <1MB) | ? (low) | ? (Linear) |
| **std::async + *default*** | In this or another thread **Enables work stealing, task inlining** | [Herb's test] Nearly constant (always <1MB) | 4 (?) | Linear, ~8usec/task (naïve attempt, didn't investigate optimizations) |