Modern DICE

Chris Hopman, Build Infra

What is DICE?

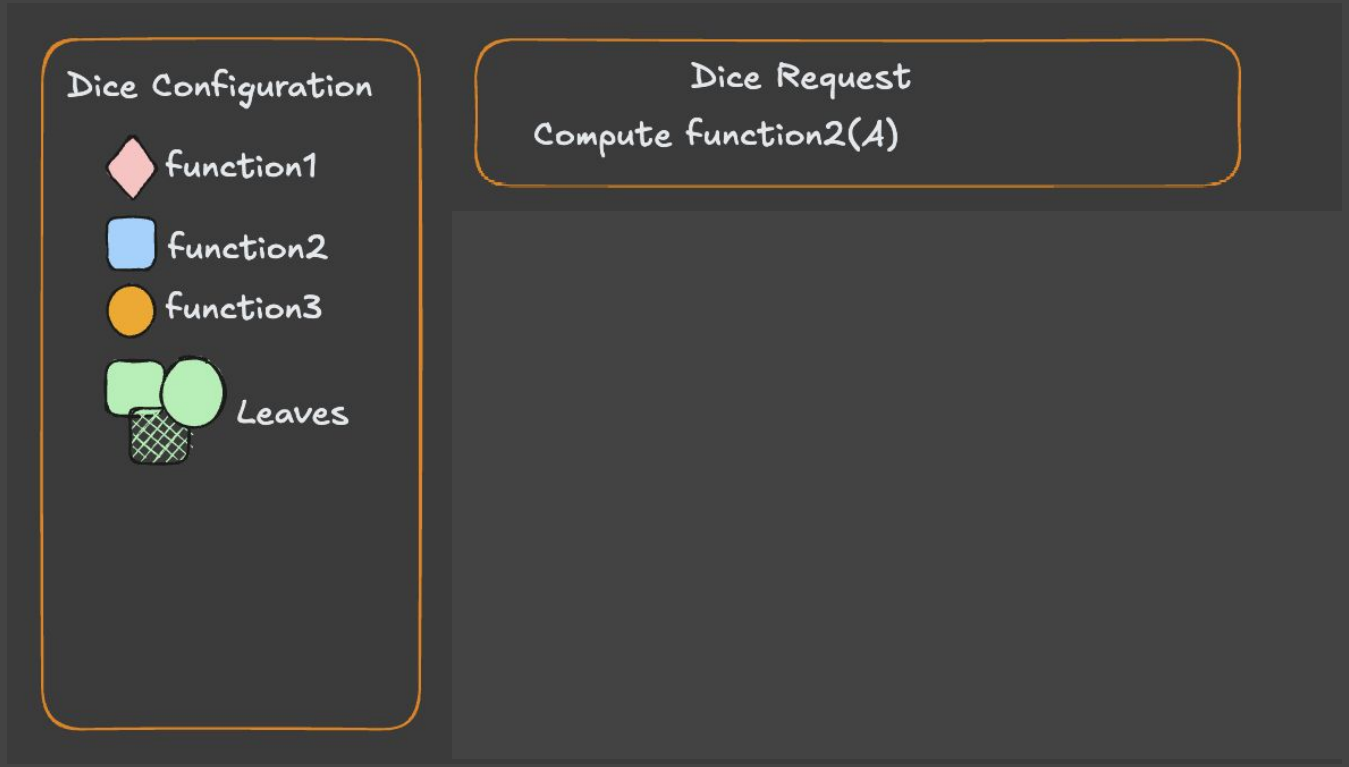
DICE: Distributed Incremental Computation Engine

What is DICE?

DICE: Distributed Incremental Computation Engine

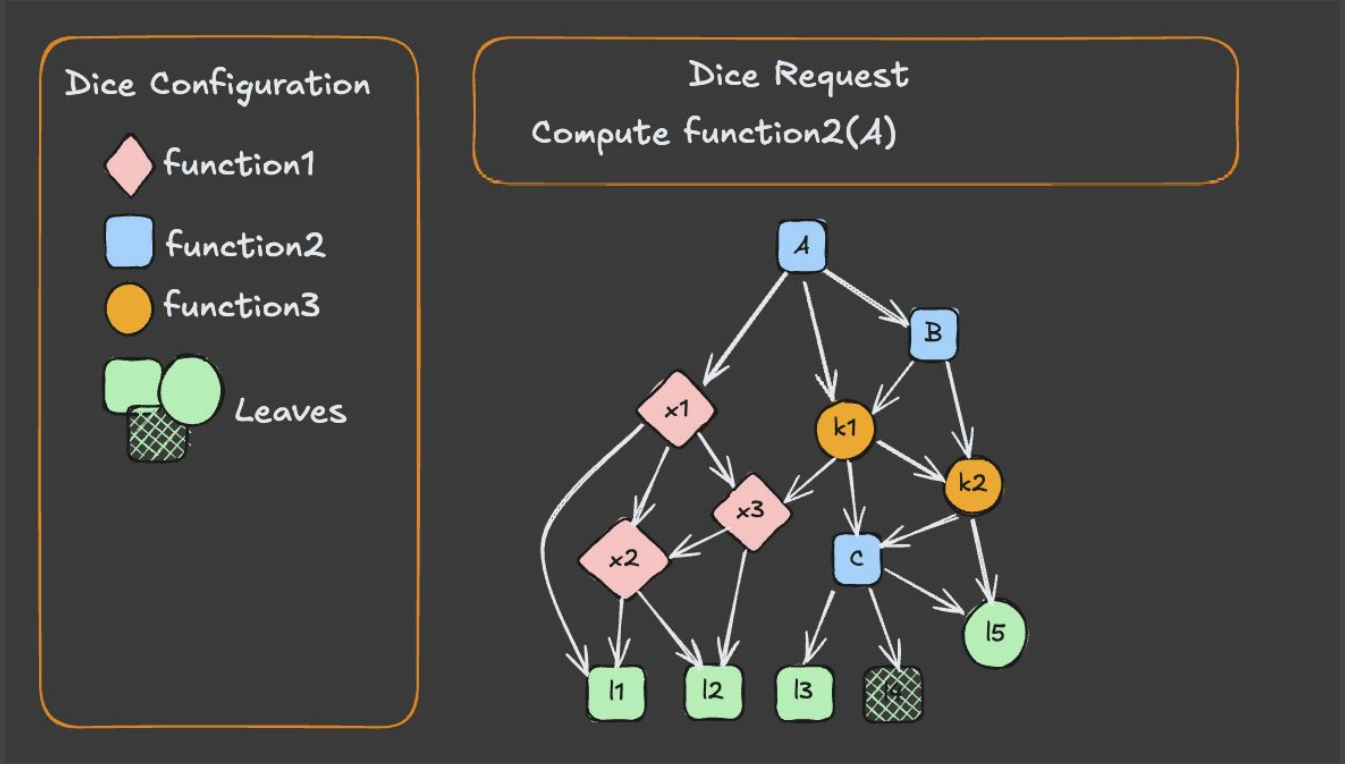
What is DICE?

DICE: Distributed Incremental Computation Engine



What is DICE?

DICE: Distributed Incremental Computation Engine

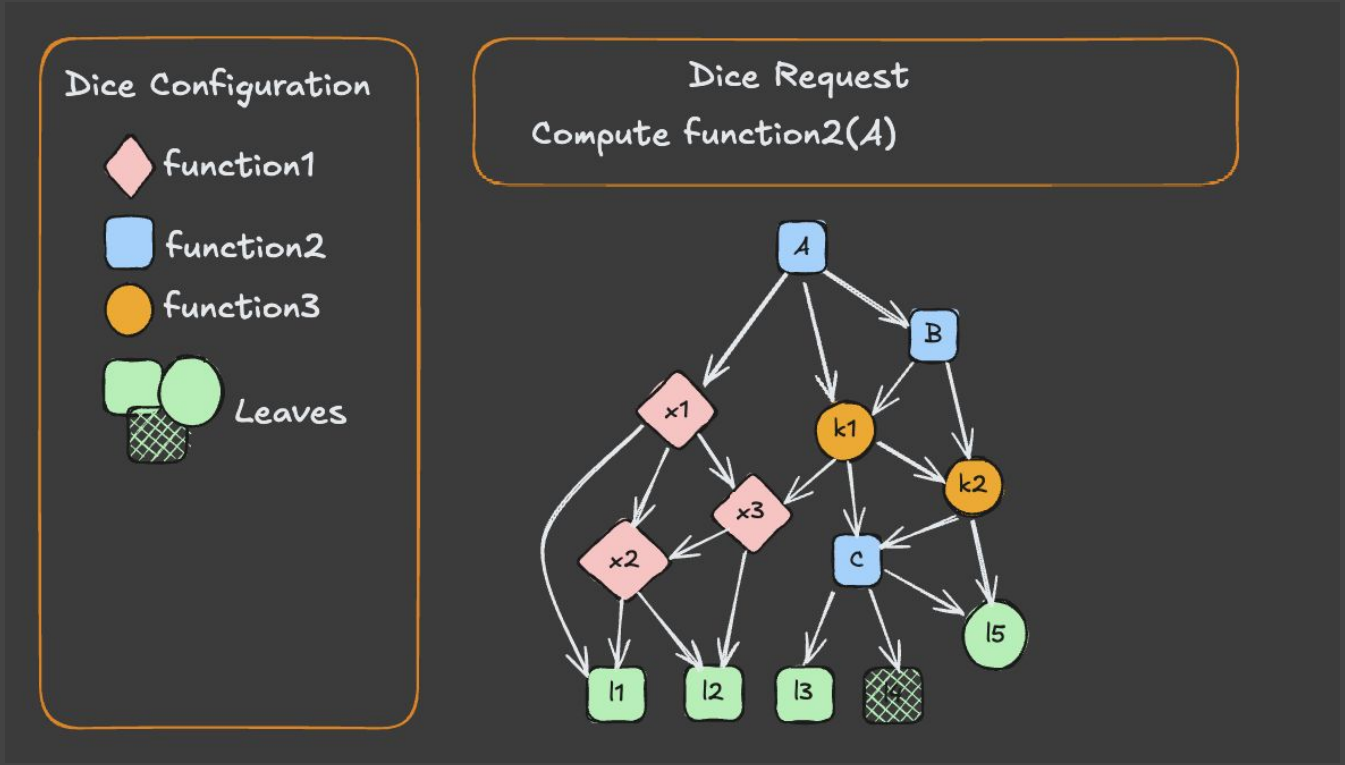


What is DICE?

DICE: Distributed Incremental Computation Engine

Computes in parallel

Automatic work sharing

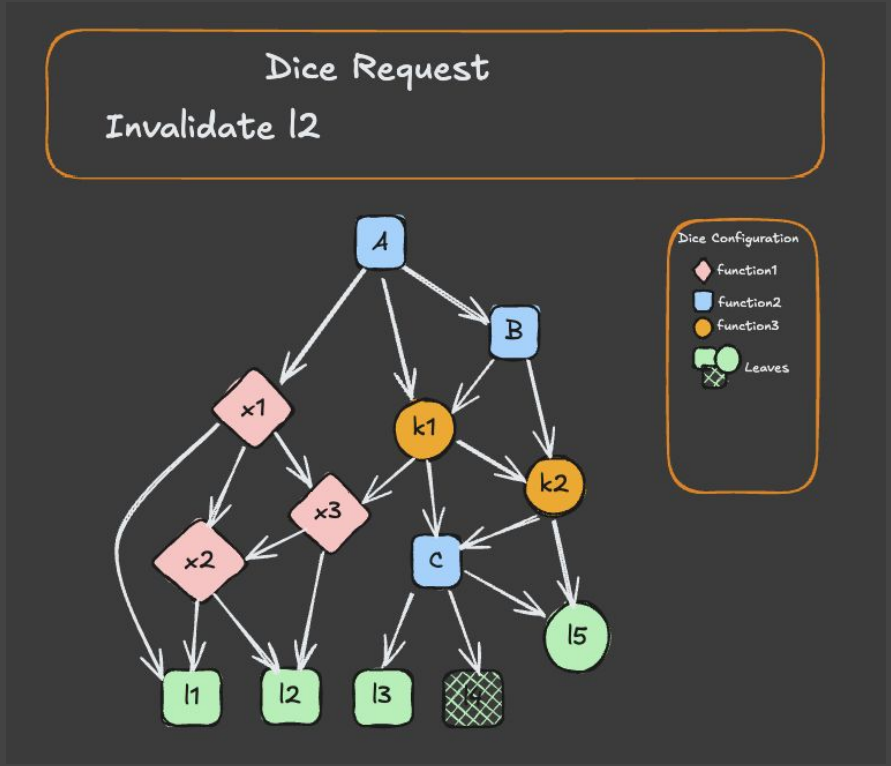


What is DICE?

DICE: Distributed Incremental Computation Engine

Computes in parallel

Automatic work sharing



What is DICE?

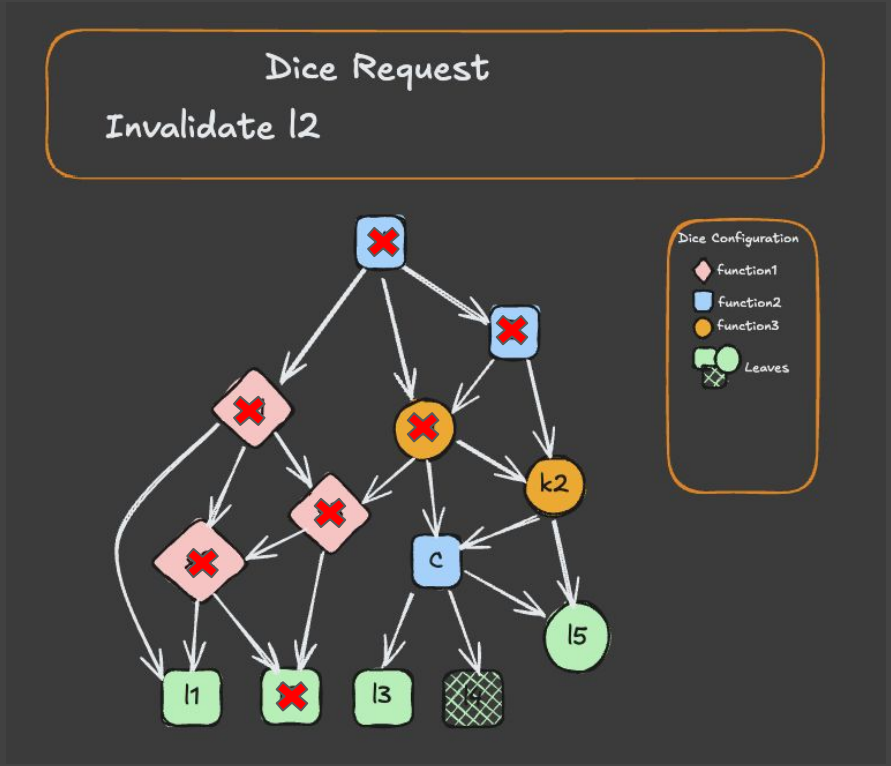
DICE: Distributed Incremental Computation Engine

Computes in parallel

Automatic work sharing

Tracks dependencies

Manages invalidation



What is DICE?

DICE: Distributed Incremental Computation Engine

Computes in parallel

Automatic work sharing

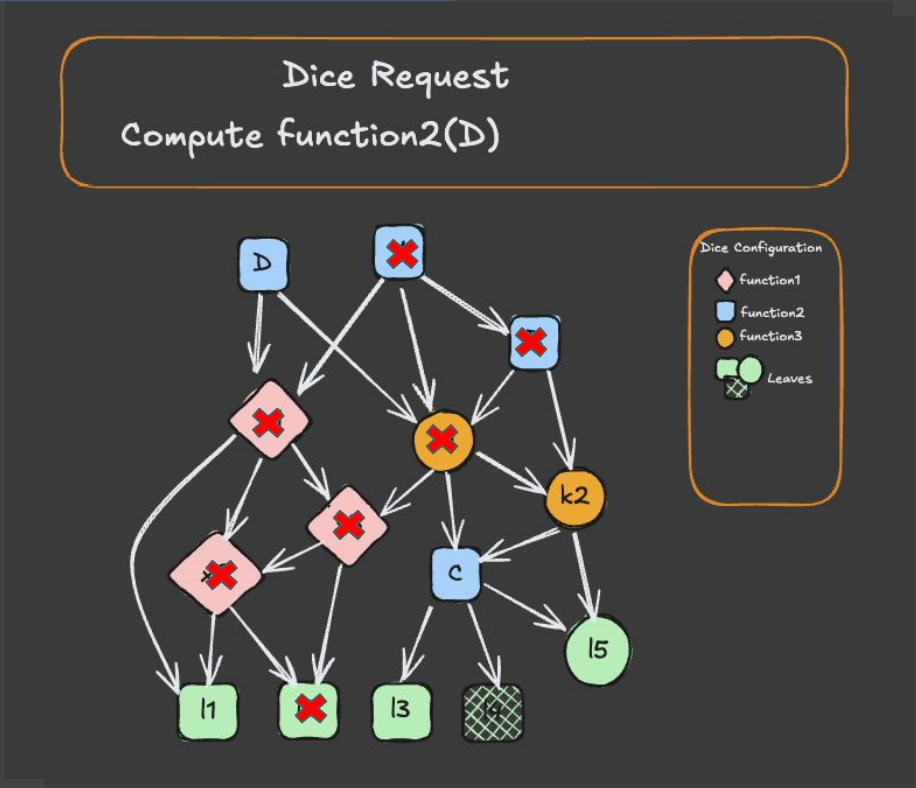
Tracks dependencies

Manages invalidation

Efficient recomputation

Early cutoff

Concurrent requests



What is DICE?

DICE: ~~Distributed~~ next time Incremental Computation Engine

Computes in parallel

Automatic work sharing

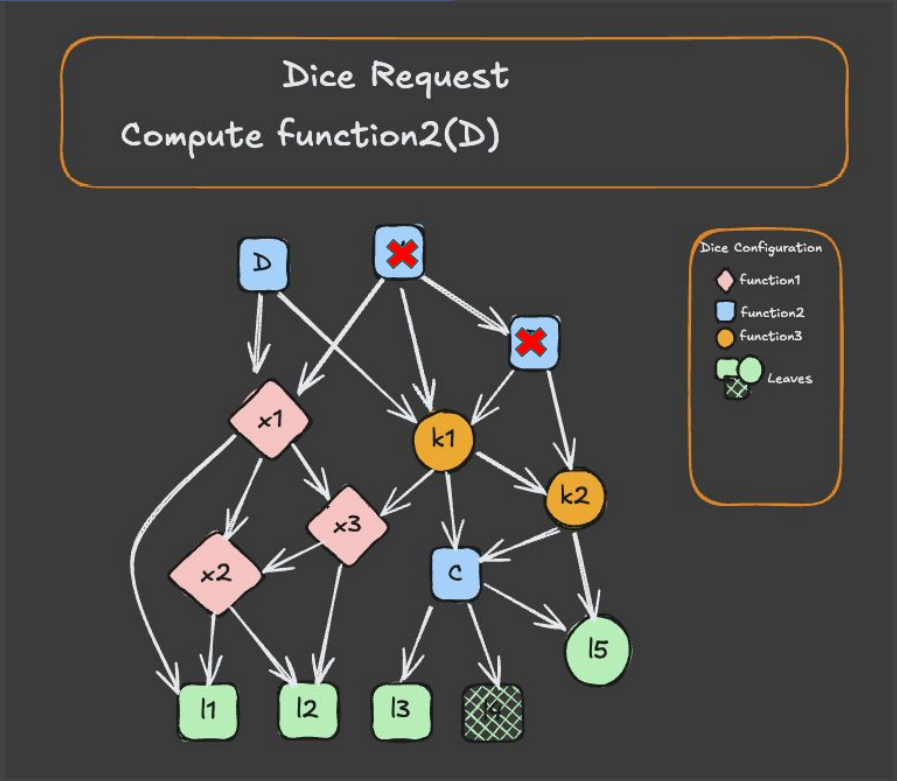
Tracks dependencies

Manages invalidation

Efficient recomputation

Early cutoff

Concurrent requests



DICE example: recursive word counts

DICE example: recursive word counts

```
async fn read_file(path: PathBuf) -> String { <...> }
async fn get_word_count(data: String) -> WordCount { <...> }
async fn list_dir(path: PathBuf) -> Vec<DirEntry> { <...> }
async fn merge(wcs: &[WordCount]) -> WordCount { <...> }
```

DICE example: recursive word counts

```
async fn read_file(path: PathBuf) -> String { <...> }
async fn get_word_count(data: String) -> WordCount { <...> }
async fn list_dir(path: PathBuf) -> Vec<DirEntry> { <...> }
async fn merge(wcs: &[WordCount]) -> WordCount { <...> }
```

100ms

10ms

10ms * len(wcs)

```
foo/
    bar/
        a.txt
        b.txt
        ...
    baz/
        a.txt
        b.txt
        ...
foo2/
    a.txt
    b.txt
```

DICE example: recursive word counts

```
async fn read_file(path: PathBuf) -> String { <...> }
async fn get_word_count(data: String) -> WordCount { <...> }
async fn list_dir(path: PathBuf) -> Vec<DirEntry> { <...> }
async fn merge(wcs: &[WordCount]) -> WordCount { <...> }

async fn word_count_recursive_v1(path: PathBuf) -> WordCount {
    let mut count_futs = Vec::new();
    let mut queue = vec![path];
    while let Some(next) = queue.pop() {
        for d in list_dir(next).await {
            match d {
                DirEntry::File(f) => counts.push(spawn(get_word_count(f)));
                DirEntry::Dir(d) => queue.push(d);
            }
        }
    }
    let counts = join_all(count_futs).await;
    merge(counts).await
}
```

100ms

10ms

10ms * len(wcs)

```
foo/
    bar/
        a.txt
        b.txt
        ...
    baz/
        a.txt
        b.txt
        ...
foo2/
    a.txt
    b.txt
```

DICE example: recursive word counts

total files: 1,000,000 (at depth 5)
total directories: 100,000 (10 children per dir)
threads: 10

get_word_count: 100ms
list_dir: 10ms
merge_word_count: 1ms * len(wcs)

```
foo/
    bar/
        a.txt
        b.txt
        ...
    baz/
        a.txt
        b.txt
        ...
foo2/
    a.txt
    b.txt
```

Scenarios:
cold - first computation
add_file - add new file at a/b/c/d/e/new.txt
add_dir - add directory w/ 10 files at a/b/c/d/e/new_dir/
fix_typo - fix a typo in a/b/c/d/e/typo.txt
rename_file - rename a/b/c/d/e/old.txt to a/b/c/d/e/new.txt

	v1				
cold	11100s				
add_file	11100s				
add_dir	11100s				
fix_typo	11100s				
rename_file	11100s				

DICE example: recursive word counts

```
async fn dice_read_file(ctx: &mut DiceComputations<'_>, path: PathBuf) -> String { <...> }
async fn get_word_count(data: String) -> WordCount { <...> }

async fn dice_get_word_count(ctx: &mut DiceComputations<'_>, path: PathBuf) -> WordCount {
   pub struct WordCountKey(PathBuf);

   impl Key for WordCountKey {
       type Value = WordCount;
       async fn compute(
           &self,
           ctx: &mut DiceComputations<'_>,
           cancellations: &CancellationContext
       ) -> WordCount {
           get_word_count(dice_read_file(ctx, self.0).await).await
       }
   }

   ctx.compute(&WordCountKey(path)).await
}
```

```
foo/
    bar/
        a.txt
        b.txt
        ...
    baz/
        a.txt
        b.txt
        ...
foo2/
    a.txt
    b.txt
```

DICE example: recursive word counts

```
async fn list_dir(path: PathBuf) -> Vec<DirEntry> { <...> }
async fn merge(wcs: &[WordCount]) -> WordCount { <...> }

async fn dice_get_word_count(ctx: &mut DiceComputations<'_>, path: PathBuf) -> WordCount {...}

async fn word_count_recursive_v2(ctx: &mut DiceComputations<'_>, path: PathBuf) -> WordCount {
    let mut files = Vec::new();
    let mut queue = vec![path];
    while let Some(next) = queue.pop() {
        for d in list_dir(next).await {
            match d {
                DirEntry::File(f) => files.push(f);
                DirEntry::Dir(d) => queue.push(d);
            }
        }
    }
    let counts = ctx.join_all(files, |ctx, f| dice_get_word_count(ctx, f)).await;
    merge(counts).await
}
```

```
foo/
    bar/
        a.txt
        b.txt
        ...
    baz/
        a.txt
        b.txt
        ...
foo2/
    a.txt
    b.txt
```

DICE example: recursive word counts

total files: 1,000,000 (at depth 5)
total directories: 100,000 (10 children per dir)
threads: 10

get_word_count: 100ms
list_dir: 10ms
merge_word_count: 1ms * len(wcs)

```
foo/
    bar/
        a.txt
        b.txt
        ...
    baz/
        a.txt
        b.txt
        ...
foo2/
    a.txt
    b.txt
```

Scenarios:
cold - first computation
add_file - add new file at a/b/c/d/e/new.txt
add_dir - add directory w/ 10 files at a/b/c/d/e/new_dir/
fix_typo - fix a typo in a/b/c/d/e/typo.txt
rename_file - rename a/b/c/d/e/old.txt to a/b/c/d/e/new.txt

	v1	v2			
cold	11100s	12000s			
add_file	11100s	2000s			
add_dir	11100s	2000s			
fix_typo	11100s	2000s			
rename_file	11100s	2000s			

DICE example: recursive word counts

```
async fn dice_get_word_count(ctx: &mut DiceComputations<'_>, path: PathBuf) -> WordCount {...}
async fn list_dir(path: PathBuf) -> Vec<DirEntry> { <...> }
async fn merge(wcs: &[WordCount]) -> WordCount { <...> }

async fn word_count_recursive_v3(ctx: &mut DiceComputations<'_>, path: PathBuf) -> WordCount {
   let entries = list_dir(path).await;
   let counts = ctx.join_all(
       entries,
       |ctx, e| match e {
           DirEntry::File(f) => dice_get_word_count(f),
           DirEntry::Dir(d) => spawn(word_count_recursive_v3(d)),
       }
   ).await;
   merge(counts).await
}
```

```
foo/
    bar/
        a.txt
        b.txt
        ...
    baz/
        a.txt
        b.txt
        ...
foo2/
    a.txt
    b.txt
```

DICE example: recursive word counts

total files: 1,000,000 (at depth 5)
total directories: 100,000 (10 children per dir)
threads: 10

get_word_count: 100ms
list_dir: 10ms
merge_word_count: 1ms * len(wcs)

```
foo/
    bar/
        a.txt
        b.txt
        ...
    baz/
        a.txt
        b.txt
        ...
foo2/
    a.txt
    b.txt
```

Scenarios:
cold - first computation
add_file - add new file at a/b/c/d/e/new.txt
add_dir - add directory w/ 10 files at a/b/c/d/e/new_dir/
fix_typo - fix a typo in a/b/c/d/e/typo.txt
rename_file - rename a/b/c/d/e/old.txt to a/b/c/d/e/new.txt

	v1	v2	v3		
cold	11100s	12000s	11100s		
add_file	11100s	2000s	1101s		
add_dir	11100s	2000s	1101s		
fix_typo	11100s	2000s	1101s		
rename_file	11100s	2000s	1101s		

DICE example: recursive word counts

```
async fn dice_word_count_recursive(ctx: &mut DiceComputations<'_>, path: PathBuf) -> WordCount {
    pub struct RecursiveWordCountKey(PathBuf);

    impl Key for RecursiveWordCountKey {
        type Value = WordCount;
        async fn compute(
            &self,
            ctx: &mut DiceComputations<'_>,
            cancellations: &CancellationContext
        ) -> WordCount {
            word_count_recursive_v4(ctx, self.0).await
        }
    }

    ctx.compute(&RecursiveWordCountKey(path)).await
}
```

```
foo/
    bar/
        a.txt
        b.txt
        ...
    baz/
        a.txt
        b.txt
        ...
foo2/
    a.txt
    b.txt
```

DICE example: recursive word counts

```
async fn dice_get_word_count(ctx: &mut DiceComputations<'_>, path: PathBuf) -> WordCount {...}
async fn merge(wcs: &[WordCount]) -> WordCount { <...> }

async fn dice_list_dir(ctx: &mut DiceComputations<'_>, path: PathBuf) -> Vec<DirEntry> { <...> }
async fn dice_word_count_recursive(ctx: &mut DiceComputations<'_>, path: PathBuf) -> WordCount {...}
```

```
async fn word_count_recursive_v4(ctx: &mut DiceComputations<'_>, path: PathBuf) -> WordCount {
    let entries = dice_list_dir(ctx, path).await;
    let counts = ctx.join_all(
        entries,
        |ctx, e| match e {
            DirEntry::File(f) => dice_get_word_count(f),
            DirEntry::Dir(d) => dice_word_count_recursive(ctx, d)
        }
    ).await;
    merge(counts).await
}
```

```
foo/
    bar/
        a.txt
        b.txt
        ...
    baz/
        a.txt
        b.txt
        ...
foo2/
    a.txt
    b.txt
```

DICE example: recursive word counts

```
foo/
    bar/
        a.txt
        b.txt
        ...
    baz/
        a.txt
        b.txt
        ...
foo2/
    a.txt
    b.txt
```

total files: 1,000,000 (at depth 5)
total directories: 100,000 (10 children per dir)
threads: 10

get_word_count: 100ms
list_dir: 10ms
merge_word_count: 1ms * len(wcs)

Scenarios:
cold - first computation
add_file - add new file at a/b/c/d/e/new.txt
add_dir - add directory w/ 10 files at a/b/c/d/e/new_dir/
fix_typo - fix a typo in a/b/c/d/e/typo.txt
rename_file - rename a/b/c/d/e/old.txt to a/b/c/d/e/new.txt

	v1	v2	v3	v4	
cold	11100s	12000s	11100s	11100s	
add_file	11100s	2000s	1101s	150ms	
add_dir	11100s	2000s	1101s	150ms	
fix_typo	11100s	2000s	1101s	150ms	
rename_file	11100s	2000s	1101s	150ms	

DICE example: recursive word counts

```
impl Key for WordCountKey {
    type Value = WordCount;
    async fn compute(...) -> WordCount {...}
    fn equality(left: &WordCount, right: &WordCount) -> bool {
        left == right
    }
}

impl Key for RecursiveWordCountKey {
    type Value = WordCount;
    async fn compute(...) -> WordCount {...}
    fn equality(left: &WordCount, right: &WordCount) -> bool {
        left == right
    }
}
```

```
foo/
    bar/
        a.txt
        b.txt
        ...
    baz/
        a.txt
        b.txt
        ...
foo2/
    a.txt
    b.txt
```

DICE example: recursive word counts

total files: 1,000,000 (at depth 5)
total directories: 100,000 (10 children per dir)
threads: 10

get_word_count: 100ms
list_dir: 10ms
merge_word_count: 1ms * len(wcs)

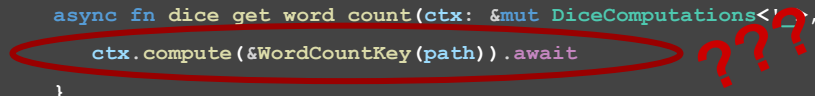
```
foo/
    bar/
        a.txt
        b.txt
        ...
    baz/
        a.txt
        b.txt
        ...
foo2/
    a.txt
    b.txt
```

Scenarios:
cold - first computation
add_file - add new file at a/b/c/d/e/new.txt
add_dir - add directory w/ 10 files at a/b/c/d/e/new_dir/
fix_typo - fix a typo in a/b/c/d/e/typo.txt
rename_file - rename a/b/c/d/e/old.txt to a/b/c/d/e/new.txt

	v1	v2	v3	v4	v5
cold	11100s	12000s	11100s	11100s	11100s
add_file	11100s	2000s	1101s	150ms	150ms
add_dir	11100s	2000s	1101s	150ms	150ms
fix_typo	11100s	2000s	1101s	150ms	100ms
rename_file	11100s	2000s	1101s	150ms	110ms

How it works: Modern DICE

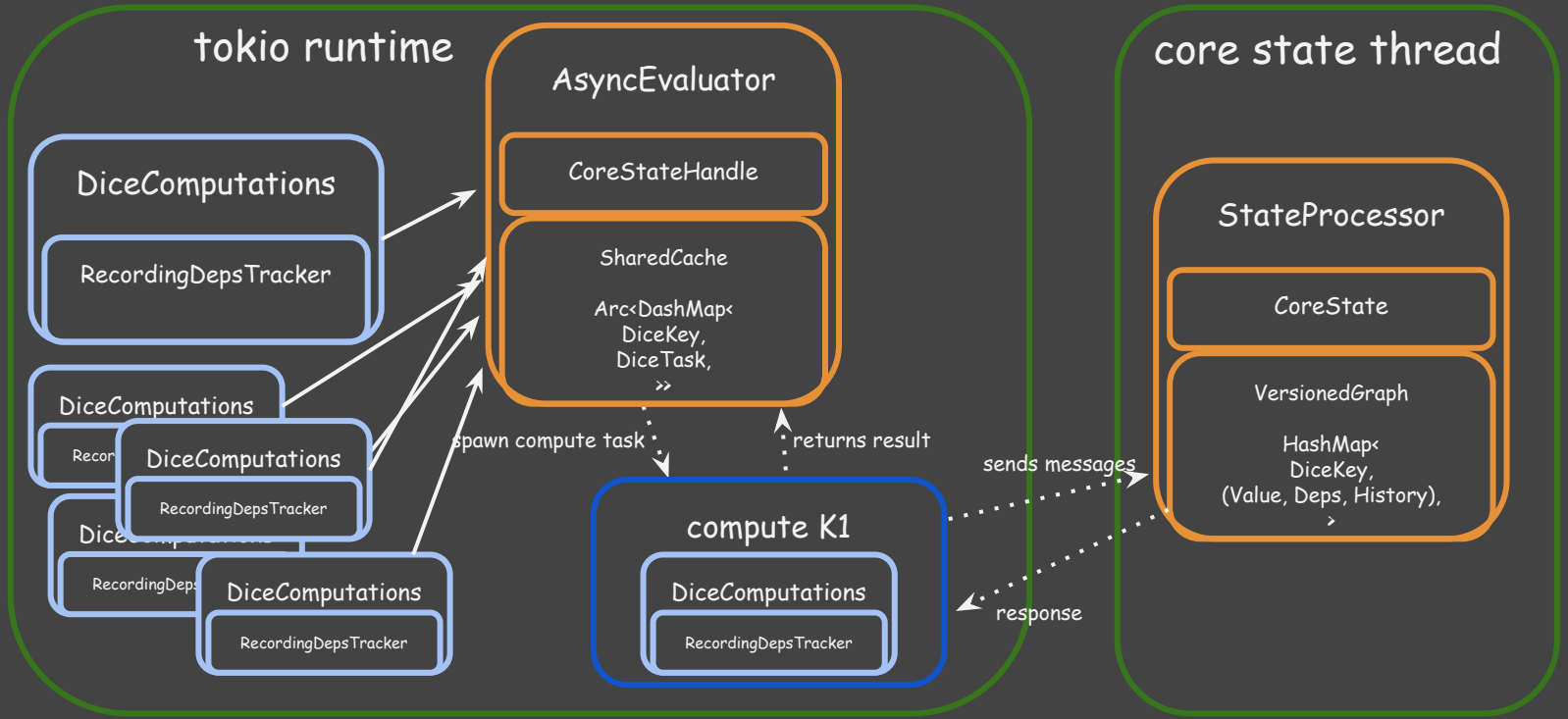
```
async fn dice_get_word_count(ctx: &mut DiceComputations<'_>, path: PathBuf) -> WordCount {
    ctx.compute(&WordCountKey(path)).await
}
```



How it works: Modern DICE

```rust
async fn dice_get_word_count(ctx: &mut DiceComputations<'_>, path: PathBuf) -> WordCount {
    ctx.compute(&WordCountKey(path)).await
}
```

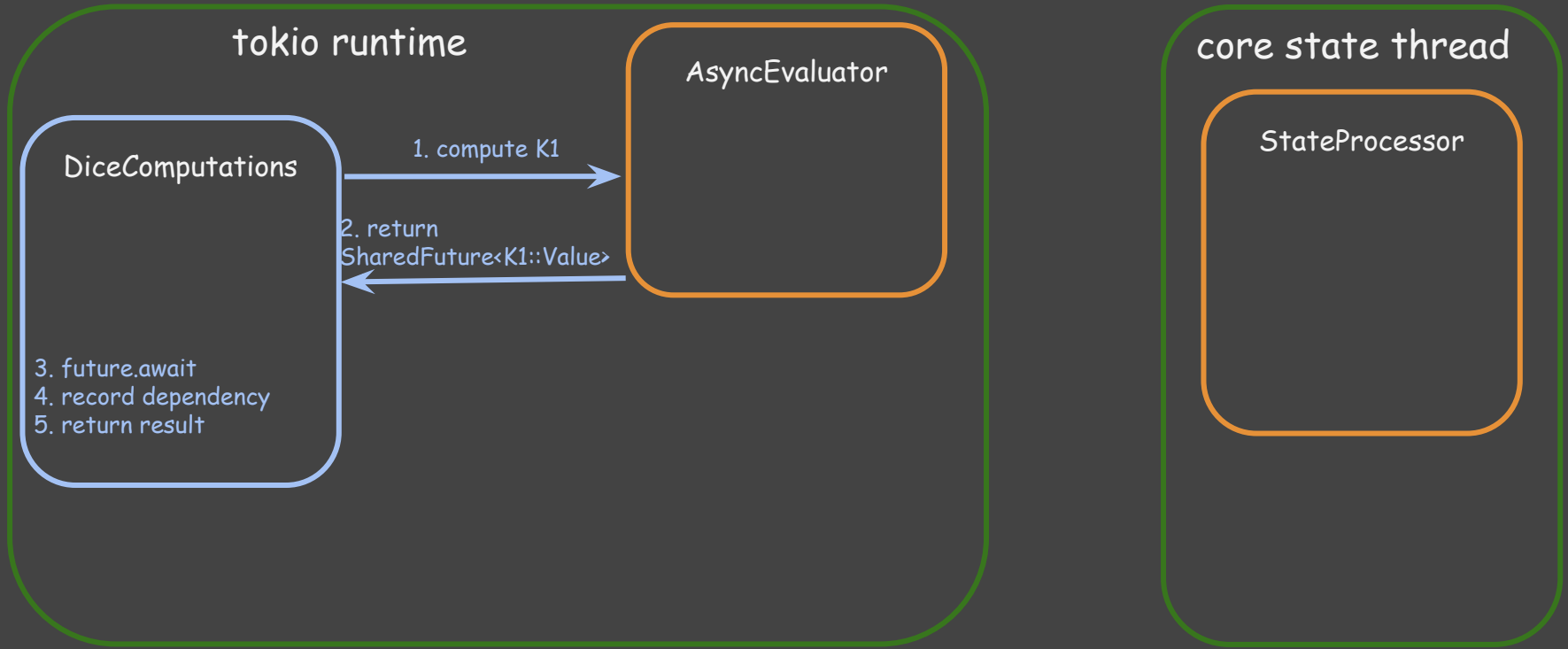
???



How it works: Modern DICE

```
async fn dice_get_word_count(ctx: &mut DiceComputations<'_>, path: PathBuf) -> WordCount {
    ctx.compute(&WordCountKey(path)).await
}
```

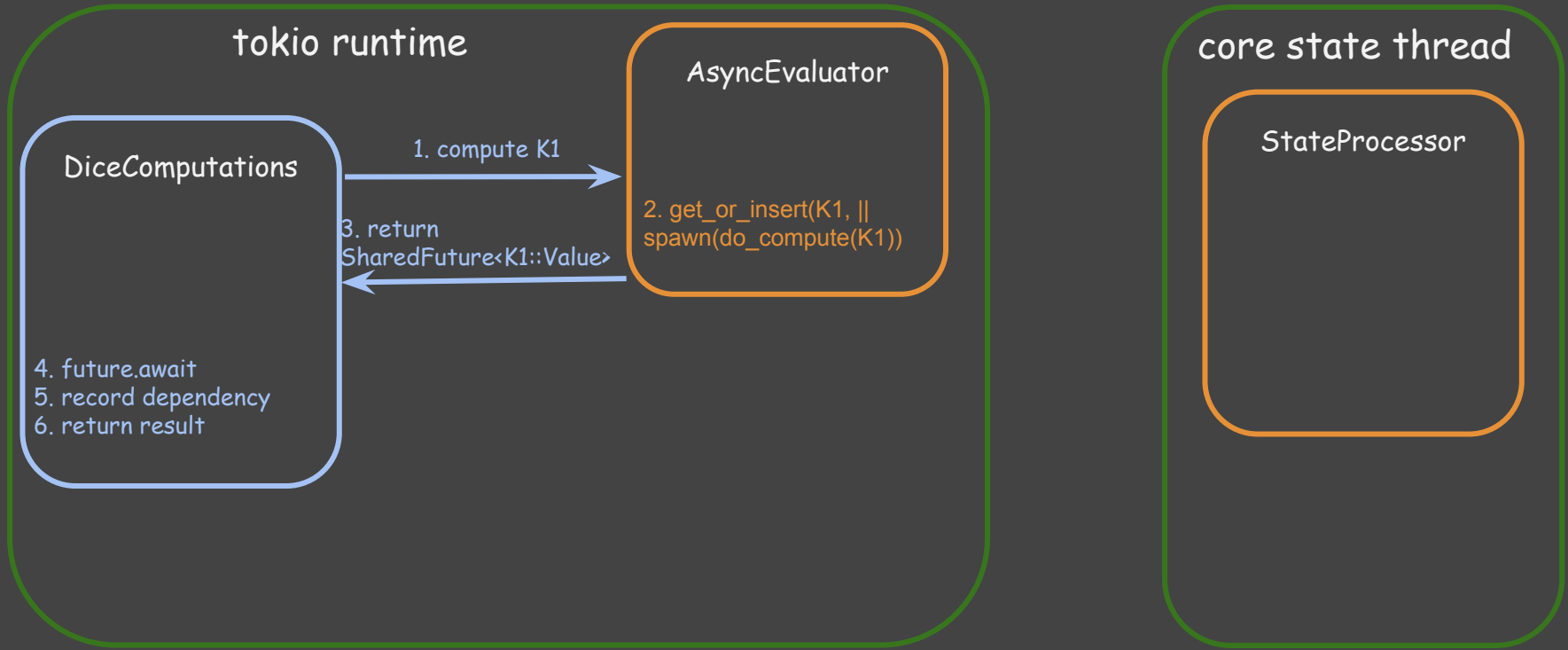
???



How it works: Modern DICE

```
async fn dice_get_word_count(ctx: &mut DiceComputations<'_>, path: PathBuf) -> WordCount {
    ctx.compute(&WordCountKey(path)).await
}
```

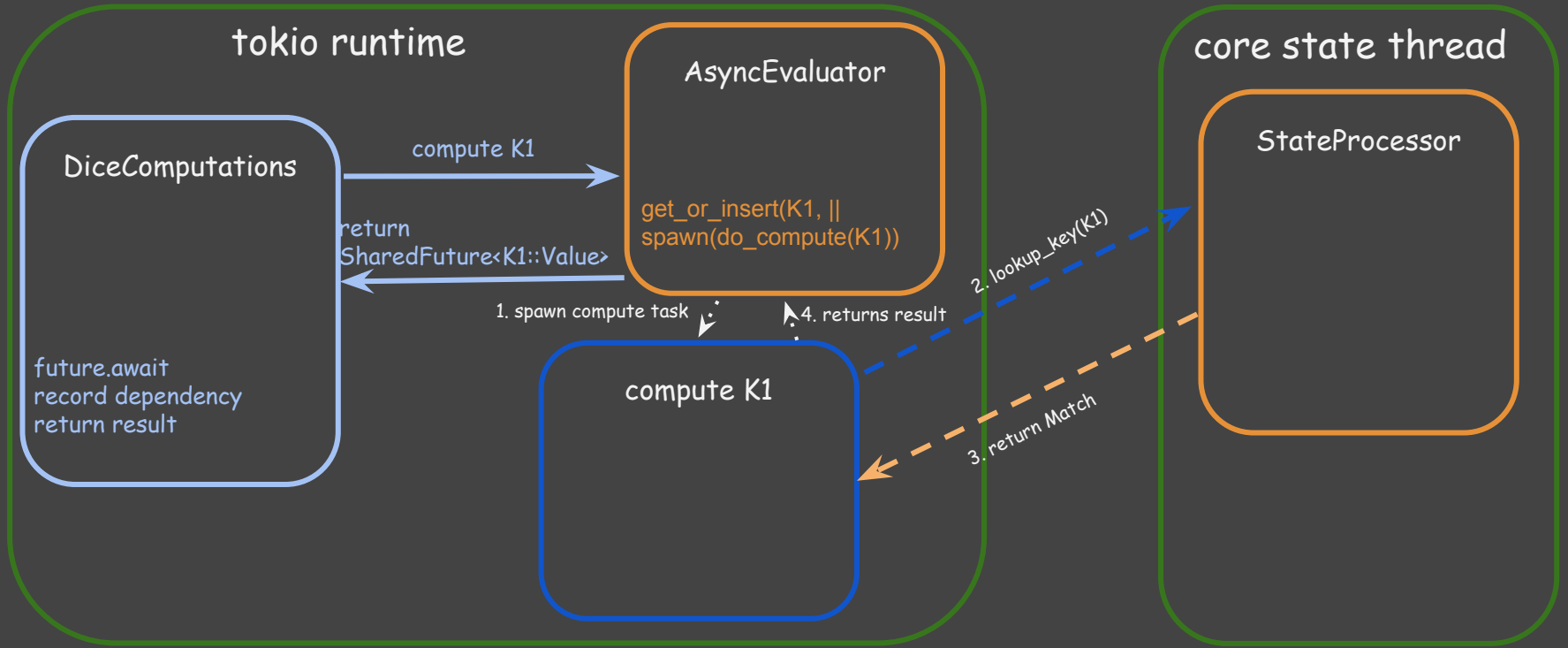
???



How it works: Modern DICE

```
async fn dice_get_word_count(ctx: &mut DiceComputations<'_>, path: PathBuf) -> WordCount {
    ctx.compute(&WordCountKey(path)).await
}
```

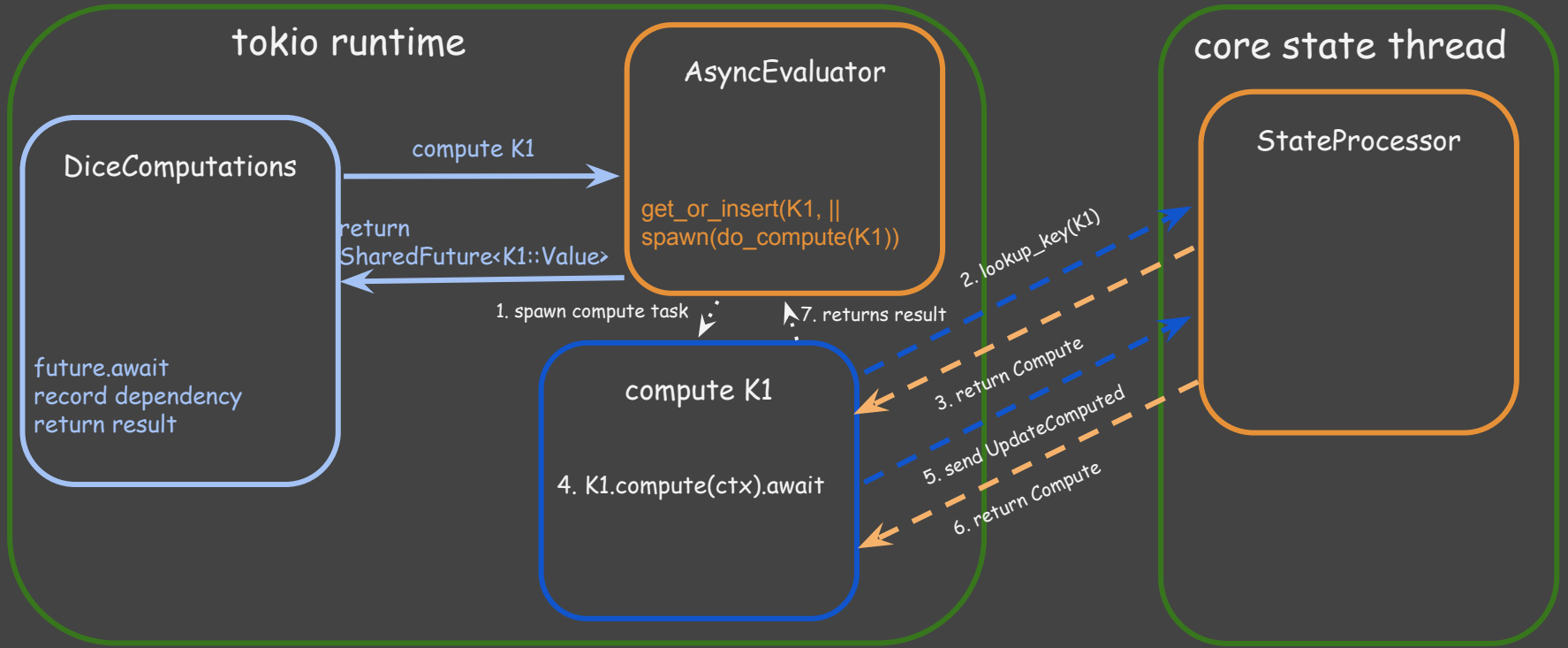
???



How it works: Modern DICE

```
async fn dice_get_word_count(ctx: &mut DiceComputations<'_>, path: PathBuf) -> WordCount {
    ctx.compute(&WordCountKey(path)).await
}
```

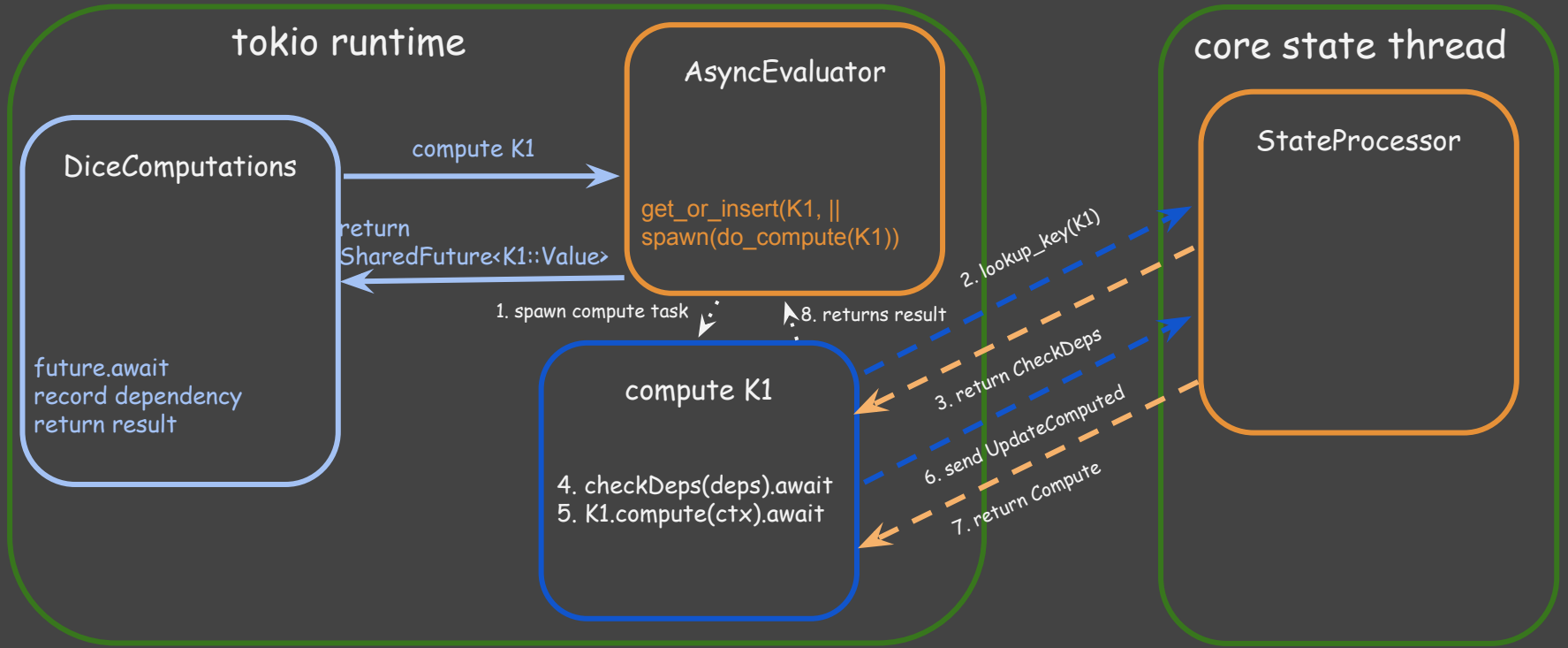
???



How it works: Modern DICE

```
async fn dice_get_word_count(ctx: &mut DiceComputations<'_>, path: PathBuf) -> WordCount {
    ctx.compute(&WordCountKey(path)).await
}
```

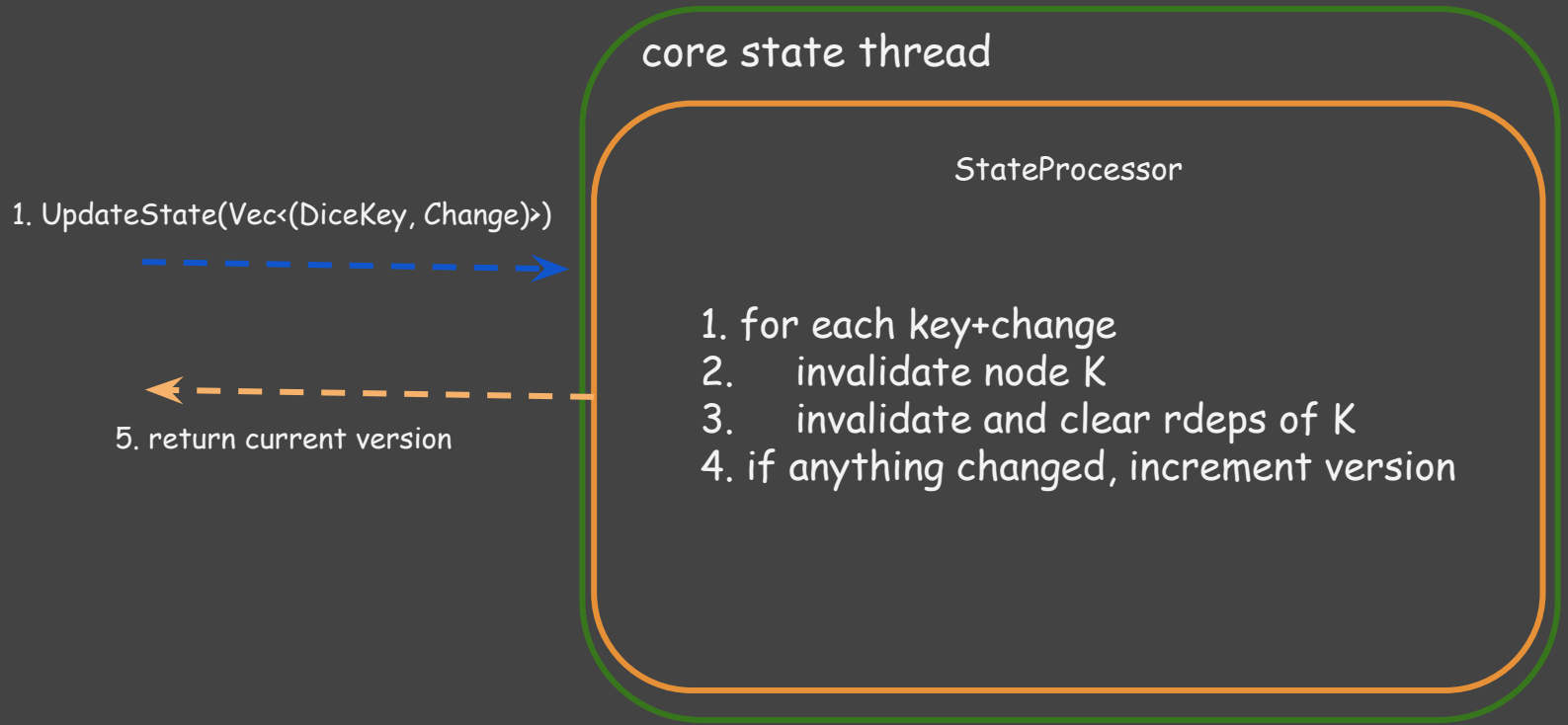
???



How it works: Modern DICE

```
async fn dice_get_word_count(ctx: &mut DiceComputations<'_>, path: PathBuf) -> WordCount {
    ctx.compute(&WordCountKey(path)).await
}
```

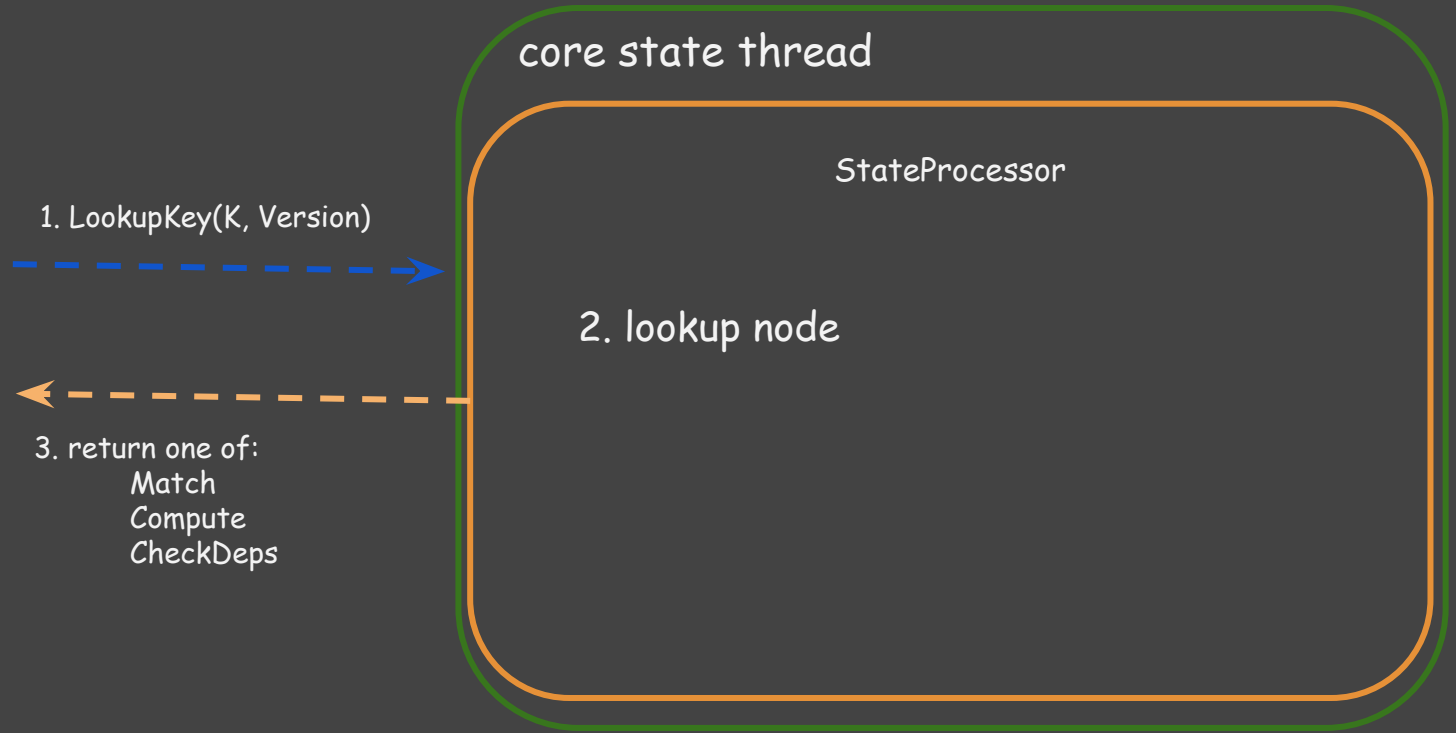
???



How it works: Modern DICE

```
async fn dice_get_word_count(ctx: &mut DiceComputations<'_>, path: PathBuf) -> WordCount {
    ctx.compute(&WordCountKey(path)).await
}
```

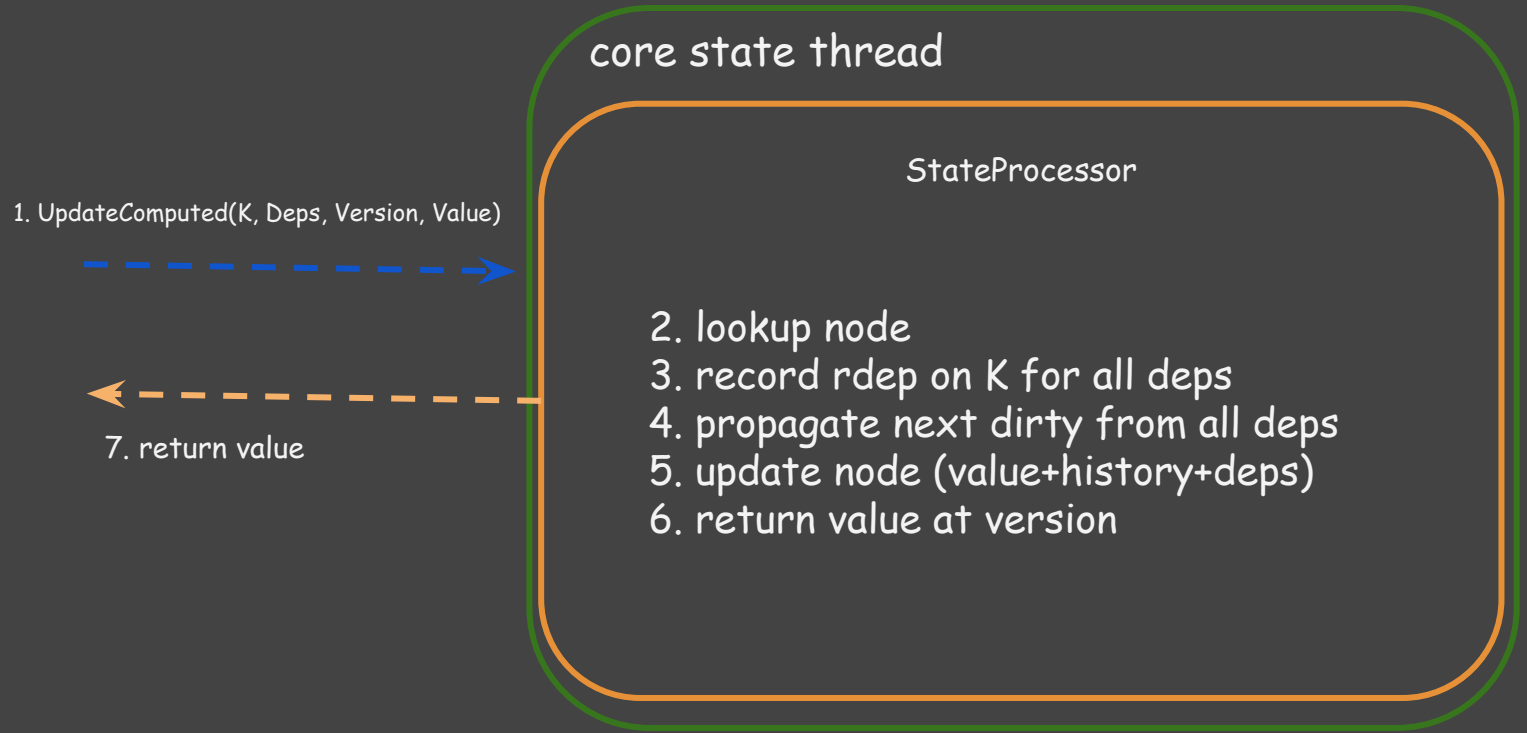
???



How it works: Modern DICE

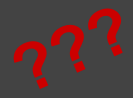
```
async fn dice_get_word_count(ctx: &mut DiceComputations<'_>, path: PathBuf) -> WordCount {
    ctx.compute(&WordCountKey(path)).await
}
```

???



How it works: Modern DICE

4. checkDeps(deps).await



Example deps for RecursiveWordCountKey(//buck2):

- ReadDirKey(//buck2)
- RecursiveWordCountKey(//buck2/dice)
- RecursiveWordCountKey(//buck2/app)
- WordCountKey(//buck2/TARGETS)
- WordCountKey(//buck2/HACKING.md)

How it works: Modern DICE

4. checkDeps(deps).await

???

Example deps for RecursiveWordCountKey(//buck2):
- ReadDirKey(//buck2)
- RecursiveWordCountKey(//buck2/dice)
- RecursiveWordCountKey(//buck2/app)
- WordCountKey(//buck2/TARGETS)
- WordCountKey(//buck2/HACKING.md)

```
async fn check_deps_v1(ctx: &mut _, deps: Vec<DiceKey>) -> CheckDepsResult {
    let deps_results = join_all(deps.iter().map(|v| check_dep(v))).await;
    CheckDepsResults::from(deps_results)
}
```

How it works: Modern DICE

4. checkDeps(deps).await

???

Example deps for RecursiveWordCountKey(//buck2):
- ReadDirKey(//buck2)
- RecursiveWordCountKey(//buck2/dice)
- RecursiveWordCountKey(//buck2/app)
- WordCountKey(//buck2/TARGETS)
- WordCountKey(//buck2/HACKING.md)

```
async fn check_deps_v1(ctx: &mut _, deps: Vec<DiceKey>) -> CheckDepsResult {
    let deps_results = join_all(deps.iter().map(|v| check_dep(v))).await;
    CheckDepsResults::from(deps_results)
}
```

Fails: panic!()

If //buck2/HACKING.md is deleted
check_deps_v1 eagerly starts compute of all deps
real compute wouldn't request //buck2/HACKING.md
check_deps_v1 panics on missing file

How it works: Modern DICE

4. checkDeps(deps).await

???

Example deps for RecursiveWordCountKey(//buck2):
- ReadDirKey(//buck2)
- RecursiveWordCountKey(//buck2/dice)
- RecursiveWordCountKey(//buck2/app)
- WordCountKey(//buck2/TARGETS)
- WordCountKey(//buck2/HACKING.md)

```rust
async fn check_deps_v2(ctx: &mut _, deps: Vec<DiceKey>) -> CheckDepsResult {
    let mut result = CheckDepsResults::Matching;
    for dep in deps {
        result.merge(check_dep(dep).await);
    }
    result
}
```

How it works: Modern DICE

4. checkDeps(deps).await

???

Example deps for RecursiveWordCountKey(//buck2):
- ReadDirKey(//buck2)
- RecursiveWordCountKey(//buck2/dice)
- RecursiveWordCountKey(//buck2/app)
- WordCountKey(//buck2/TARGETS)
- WordCountKey(//buck2/HACKING.md)

```
async fn check_deps_v2(ctx: &mut _, deps: Vec<DiceKey>) -> CheckDepsResult {
    let mut result = CheckDepsResults::Matching;
    for dep in deps {
        result.merge(check_dep(dep).await);
    }
    result
}
```

Fails: too slow

Only triggers one dep check at a time
Leads to recomputations being single threaded

How it works: Modern DICE

4. checkDeps(deps).await

???

Example deps for RecursiveWordCountKey(//buck2):
- ReadDirKey(//buck2)
- RecursiveWordCountKey(//buck2/dice)
- RecursiveWordCountKey(//buck2/app)
- WordCountKey(//buck2/TARGETS)
- WordCountKey(//buck2/HACKING.md)

```
let counts = ctx.join_all(files, |ctx, f| dice_get_word_count(ctx, f)).await;
```

Not the same ctx!

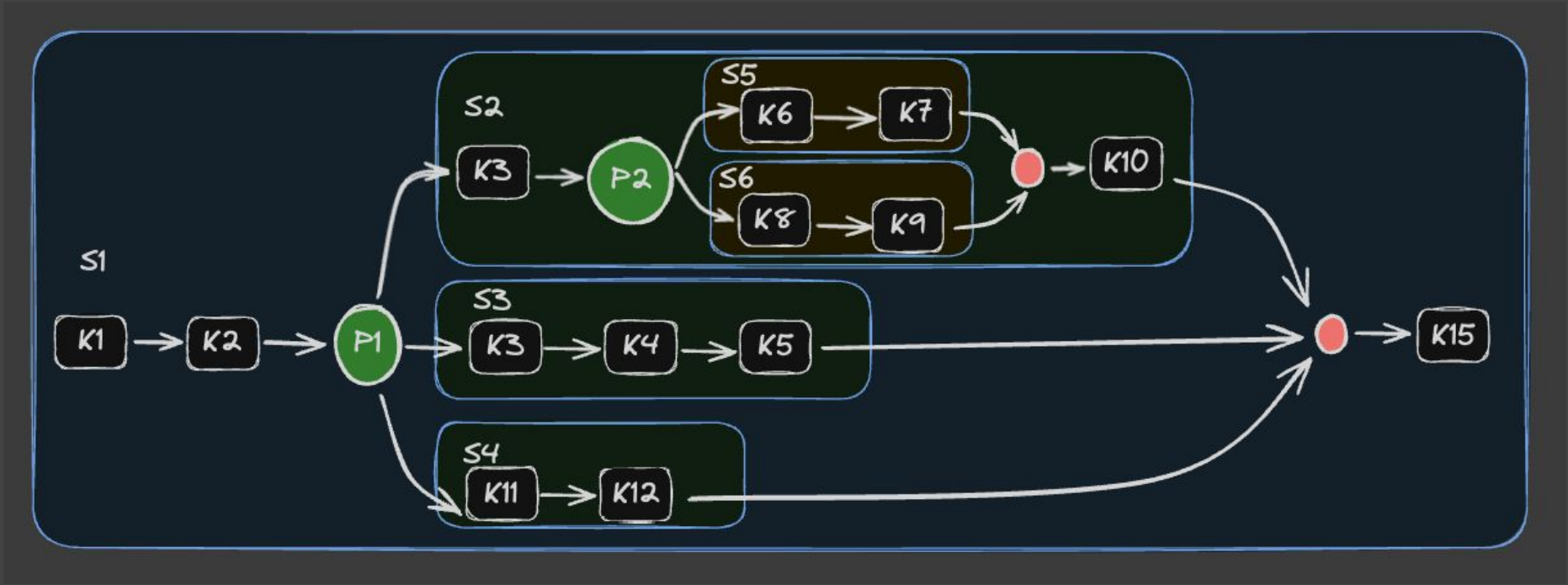
deps are recorded as a "series parallel graph"
ctx.join_all does a big trick
it maps a list into a list of futures, each of those mappings gets a different ctx
each inner ctx tracks the deps that it encounters
when the futures finish, the outer ctx creates a parallel node in its deps record with all of the inner ctx deps
rust &mut borrows help prevent non-recorded data flow

How it works: Modern DICE

4. checkDeps(deps).await

???

Example deps for RecursiveWordCountKey(//buck2):
- ReadDirKey(//buck2)
- RecursiveWordCountKey(//buck2/dice)
- RecursiveWordCountKey(//buck2/app)
- WordCountKey(//buck2/TARGETS)
- WordCountKey(//buck2/HACKING.md)



How it works: Modern DICE

4. checkDeps(deps).await

???

Example deps for RecursiveWordCountKey(//buck2):
- ReadDirKey(//buck2)
- RecursiveWordCountKey(//buck2/dice)
- RecursiveWordCountKey(//buck2/app)
- WordCountKey(//buck2/TARGETS)
- WordCountKey(//buck2/HACKING.md)

```
async fn check_deps_v3(ctx: &mut _, deps: SPGraph<DiceKey>) -> CheckDepsResult {
    let mut result = CheckDepsResults::Matching;
    for node in deps {
        if result != CheckDepsResults::Matching {
            return result;
        }
        match node {
            SPGraph::Parallel(pnodes) => {
                result.merge(join_all(pnodes.iter().map(|v| check_deps_v3(v))).await);
            }
            SPGraph::Serial(key) => {
                result.merge(check_dep(key).await);
            }
        }
    }
    result
}
```

Passes!!!

Bonus: after encountering a changed dep, parallel nodes don't need to be cancelled and can still eagerly recompute

Some additional information

How does data flow into the DICE computation? Three ways

InjectedKey - nodes where the value of the node must be set from outside dice before being computed.

- Used for global data
- Data is in only a single state for an entire transaction
- Examples: buckconfig, buck-out path, prelude path
- files and dirs act like this, but aren't technically injected

UserData - non-tracked data accessible to all dice nodes

- Supports both per-dice instance and per-dice transaction
- Should be used only for things that don't affect the computation
- Examples: EventDispatcher, remote execution client

Keys - data contained in the top-level Keys themselves

- Examples: bxl args, cli configuration modifiers

Some additional information

Some best practices

Keys:

- Try to make key values stable as things change
  - improves incrementality, prevents leaked orphaned nodes
- Equality must be correct, user's responsibility
  - Don't hide fields in keys from hash/eq
  - Need to consider equality across different states of the graph

Values:

- Design with value equality in mind, early cutoff is very powerful

Computations

- To do things in parallel, use the ctx parallel apis:
  - compute2, compute3, compute_join, and the try_ variants
- Think about the different ways that the node can be invalidated
  - consider splitting into multiple nodes for better incrementality

Other

- Do not put data that affects the results of computations in UserData
- Do not allow untracked data flow between nodes
  - Ex: a value having a mutable field (i.e. OnceLock, Mutex) could allow untracked data flow

Questions?