going just far enough with generics in bbqueue

bbqueue is
a fancy (spsc-ish) ring buffer

push and pop are
"two stage"

stage 1:
"grants" of storage space

stage 2:
"commit" or "release" the grants

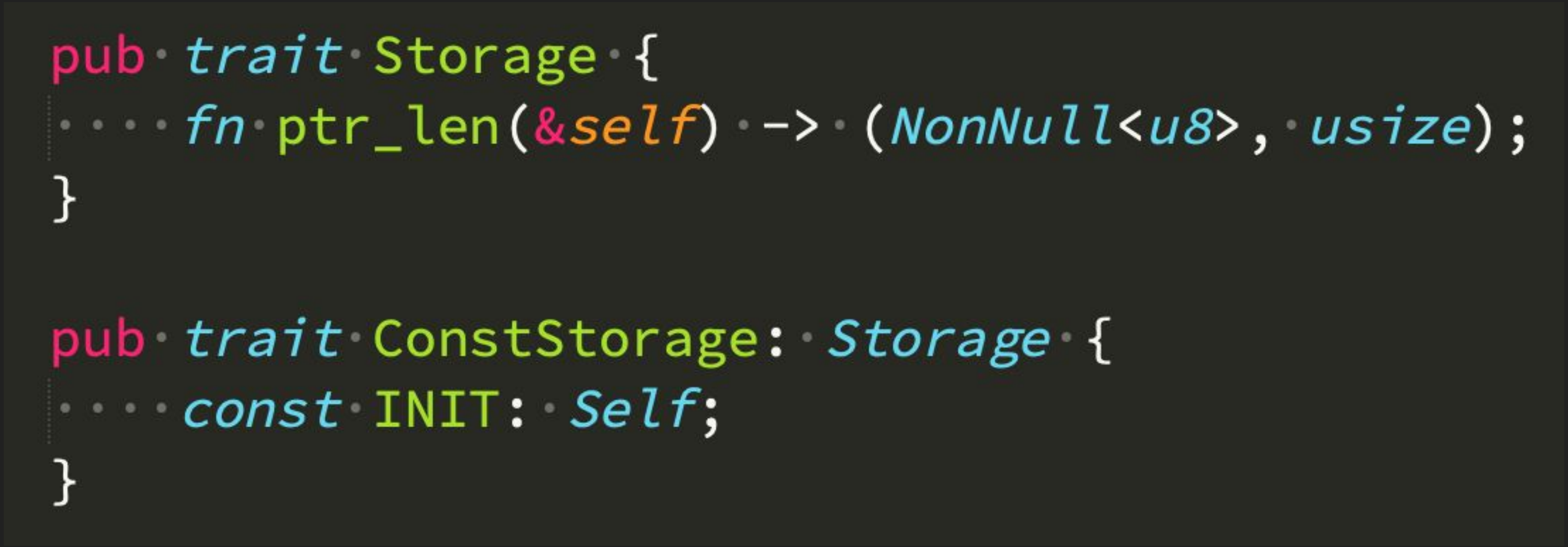
why two stages?

do things “chunk at a time”:
less overhead push + popping

grants can be a stable slice:
perfect for DMA or zero copy

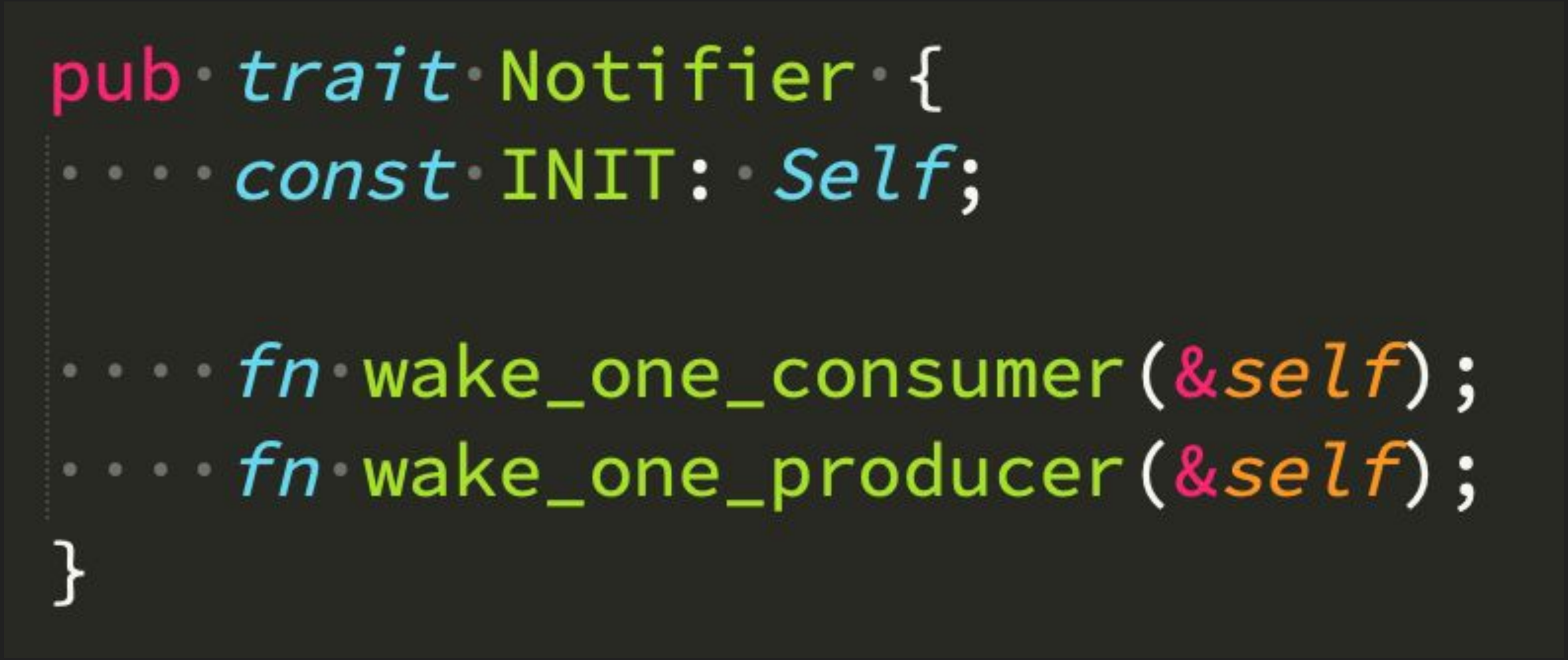
problem:
you might want to use bbqueue in a bunch of different ways.

inline or heap storage?

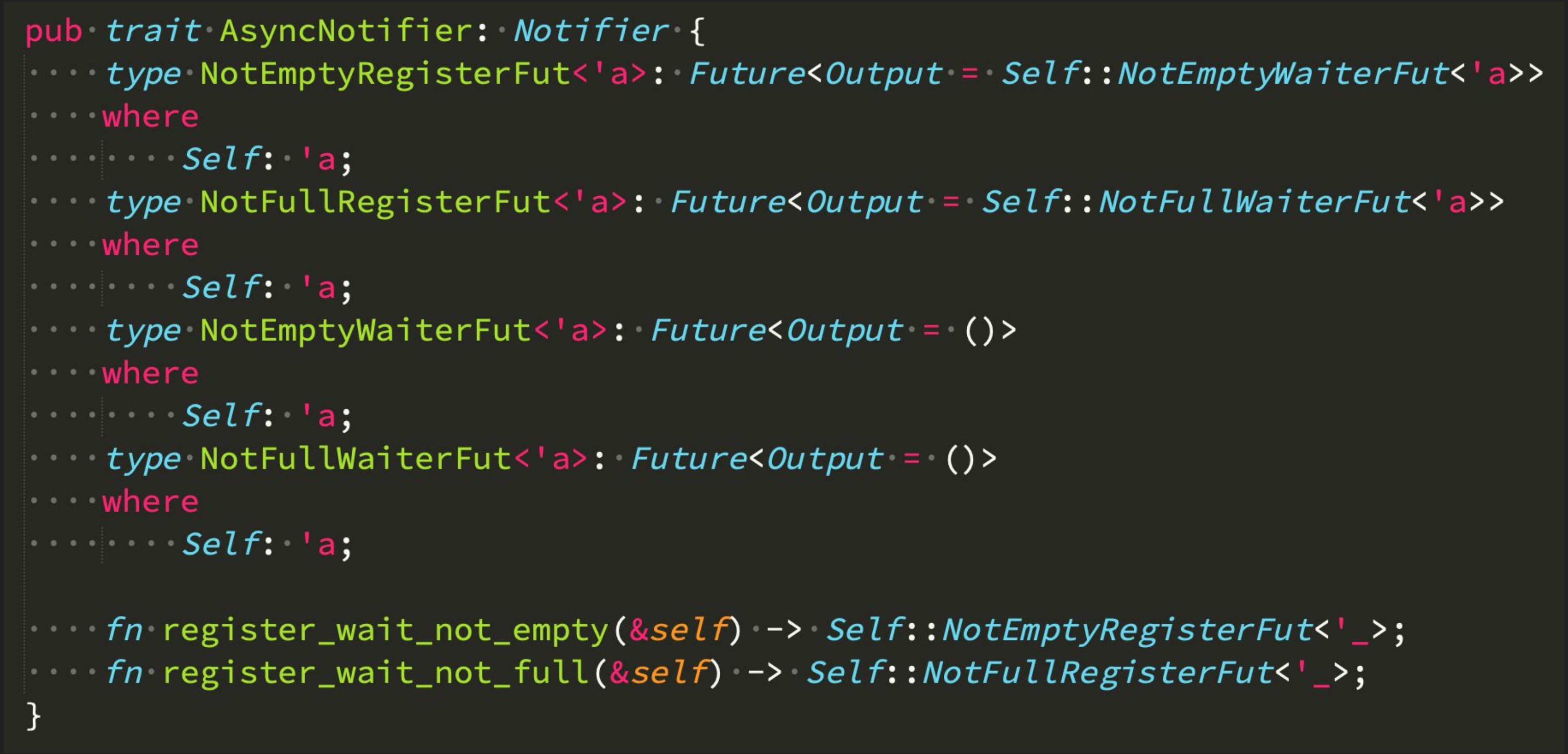


```rust
pub trait Storage {
    fn ptr_len(&self) -> (NonNull<u8>, usize);
}

pub trait ConstStorage: Storage {
    const INIT: Self;
}
```

async or not?



```rust
pub trait Notifier {
    const INIT: Self;

    fn wake_one_consumer(&self);
    fn wake_one_producer(&self);
}
```

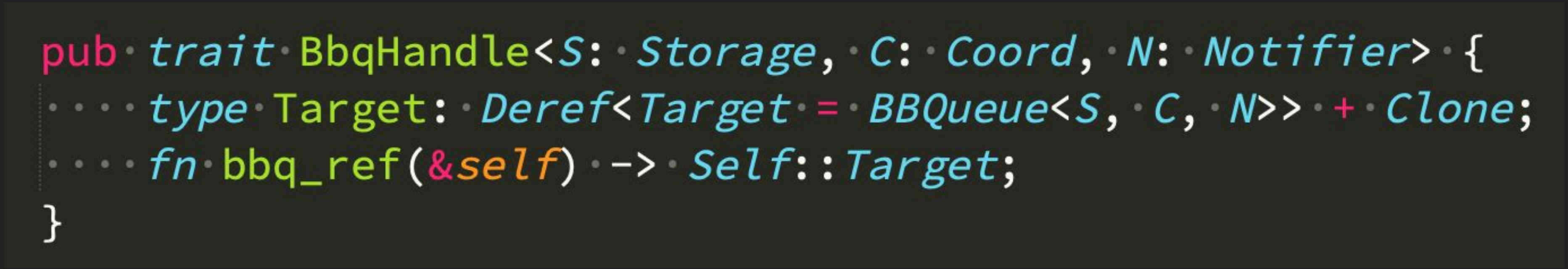


```rust
pub trait AsyncNotifier: Notifier {
    type NotEmptyRegisterFut<'a>: Future<Output = Self::NotEmptyWaiterFut<'a>>
    where
        Self: 'a;
    type NotFullRegisterFut<'a>: Future<Output = Self::NotFullWaiterFut<'a>>
    where
        Self: 'a;
    type NotEmptyWaiterFut<'a>: Future<Output = ()>
    where
        Self: 'a;
    type NotFullWaiterFut<'a>: Future<Output = ()>
    where
        Self: 'a;

    fn register_wait_not_empty(&self) -> Self::NotEmptyRegisterFut<'_>;
    fn register_wait_not_full(&self) -> Self::NotFullRegisterFut<'_>;
}
```

`Atomic` or not?

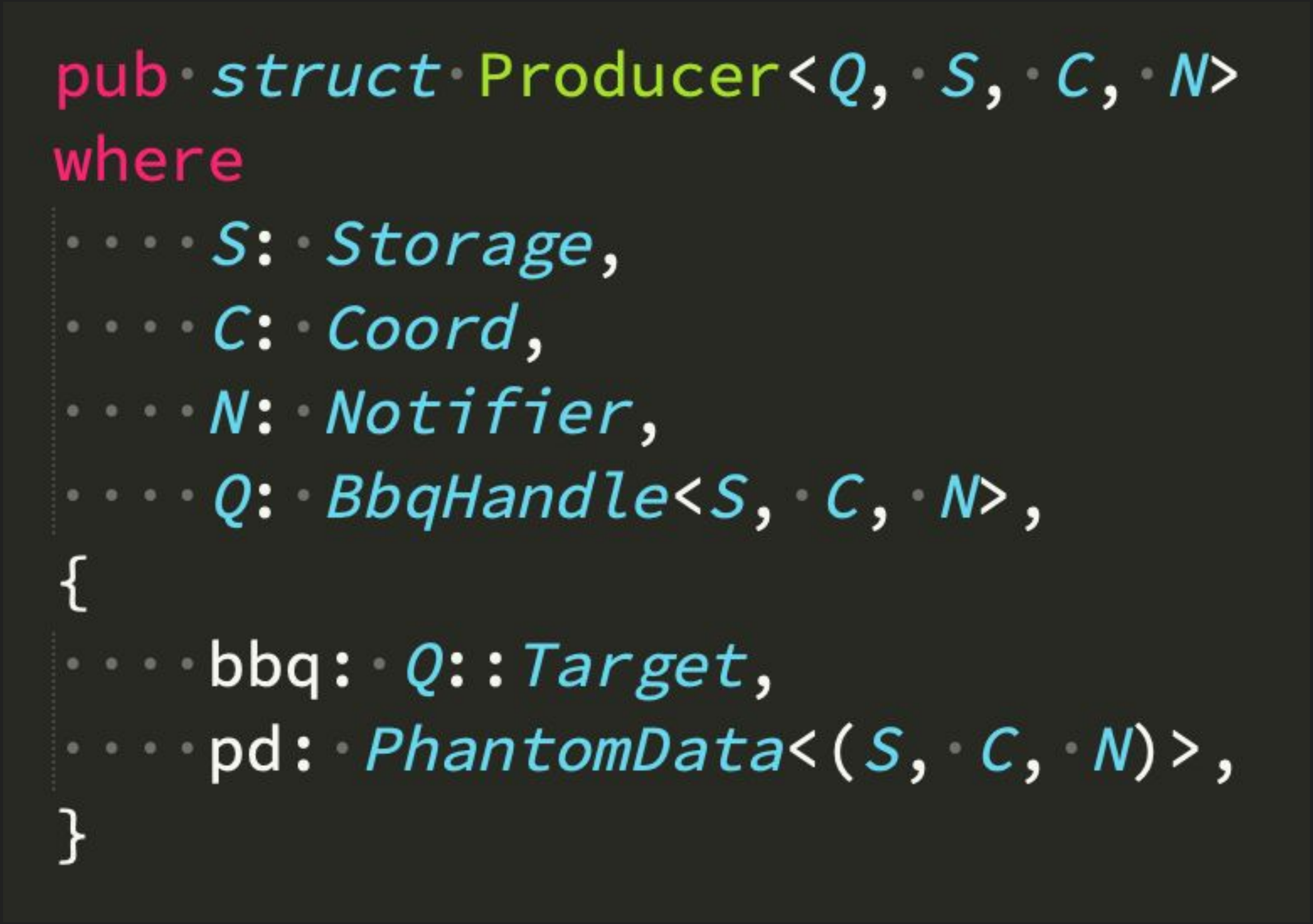
```rust
pub unsafe trait Coord {
    const INIT: Self;

    // Reset all values back to the initial empty state
    fn reset(&self);

    // Write Grants

    fn grant_max_remaining(&self, capacity: usize, sz: usize)
        -> Result<(usize, usize), ()>;
    fn grant_exact(&self, capacity: usize, sz: usize)
        -> Result<(usize, usize), ()>;
    fn commit_inner(&self, capacity: usize, grant_len: usize, used: usize);

    // Read Grants

    fn read(&self) -> Result<(usize, usize), ()>;
    fn release_inner(&self, used: usize);
}
```

borrowed or `Arc` metadata?



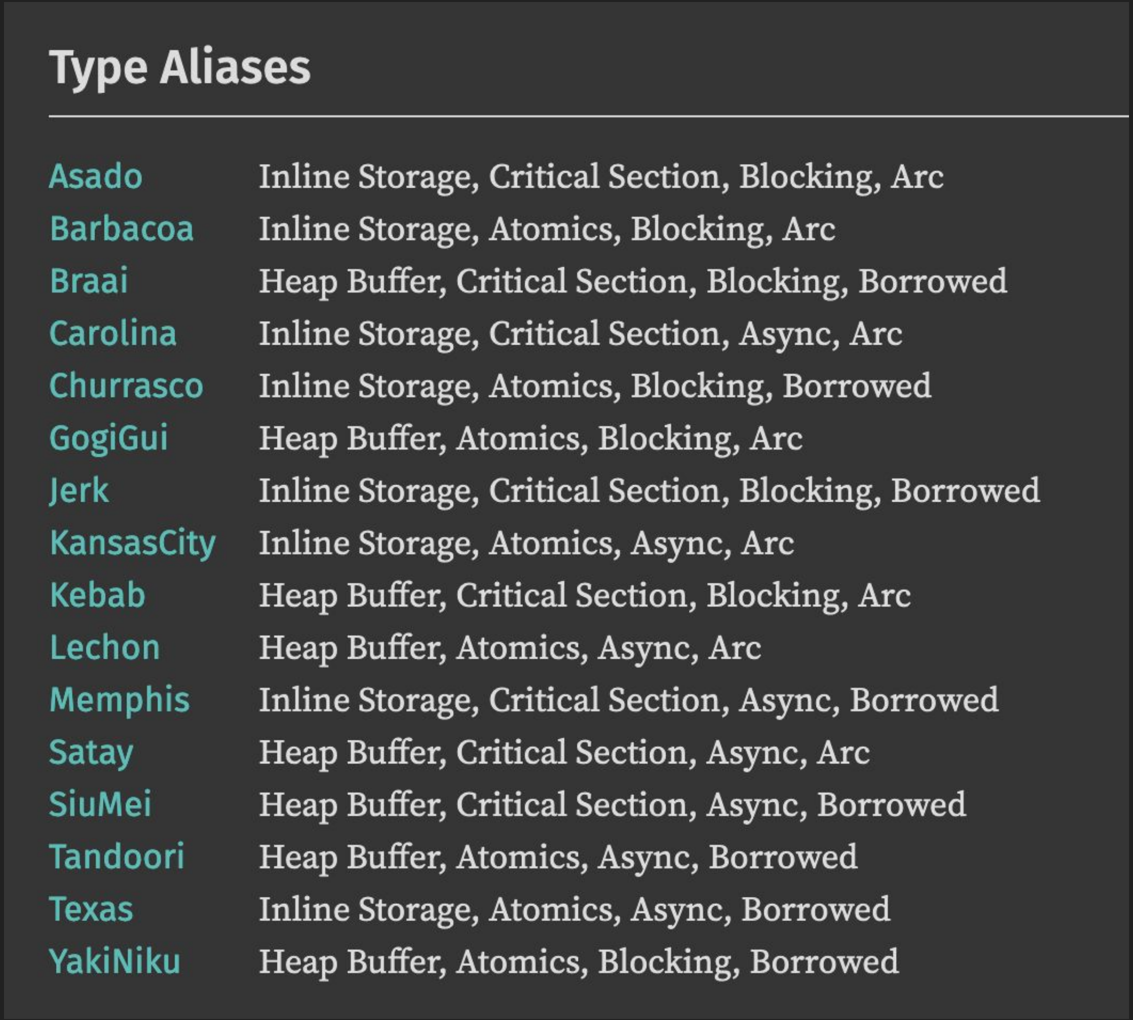
```rust
pub trait BbqHandle<S: Storage, C: Coord, N: Notifier> {
    type Target: Deref<Target = BBQueue<S, C, N>> + Clone;
    fn bbq_ref(&self) -> Self::Target;
}
```

what do you get?



```rust
pub struct Producer<Q, S, C, N>
where
    S: Storage,
    C: Coord,
    N: Notifier,
    Q: BbqHandle<S, C, N>,
{
    bbq: Q::Target,
    pd: PhantomData<(S, C, N)>,
}
```

$2^4 = 16$ combinations



Type Aliases

Asado	Inline Storage, Critical Section, Blocking, Arc
Barbacoa	Inline Storage, Atomics, Blocking, Arc
Braai	Heap Buffer, Critical Section, Blocking, Borrowed
Carolina	Inline Storage, Critical Section, Async, Arc
Churrasco	Inline Storage, Atomics, Blocking, Borrowed
GogiGui	Heap Buffer, Atomics, Blocking, Arc
Jerk	Inline Storage, Critical Section, Blocking, Borrowed
KansasCity	Inline Storage, Atomics, Async, Arc
Kebab	Heap Buffer, Critical Section, Blocking, Arc
Lechon	Heap Buffer, Atomics, Async, Arc
Memphis	Inline Storage, Critical Section, Async, Borrowed
Satay	Heap Buffer, Critical Section, Async, Arc
SiuMei	Heap Buffer, Critical Section, Async, Borrowed
Tandoori	Heap Buffer, Atomics, Async, Borrowed
Texas	Inline Storage, Atomics, Async, Borrowed
YakiNiku	Heap Buffer, Atomics, Blocking, Borrowed