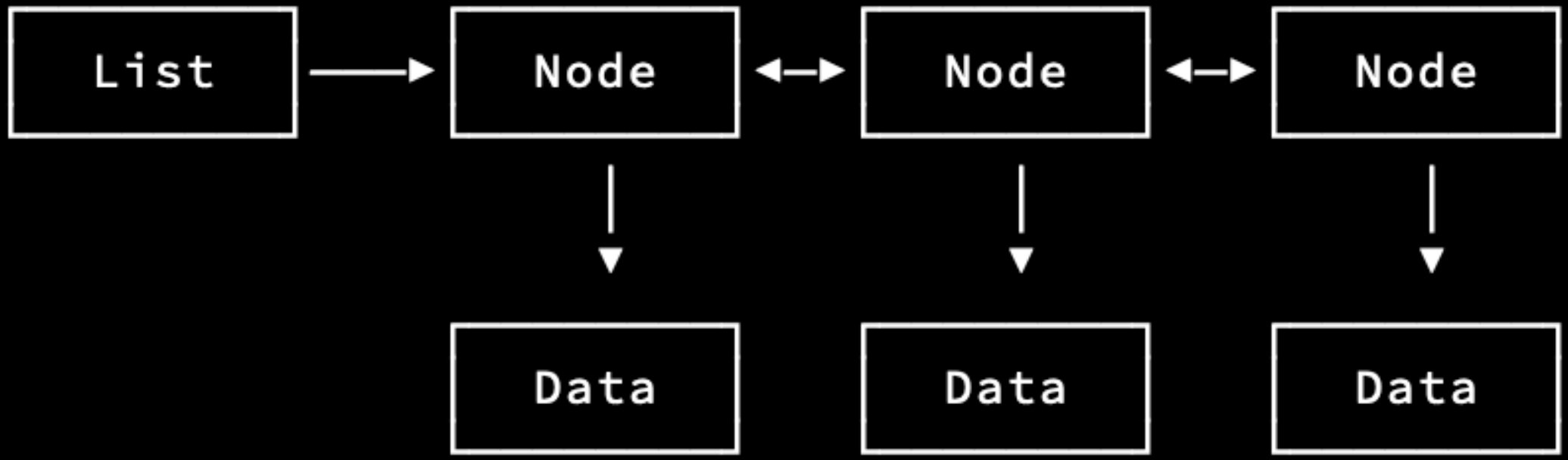
intrusive lists for fun and profit (on embedded)

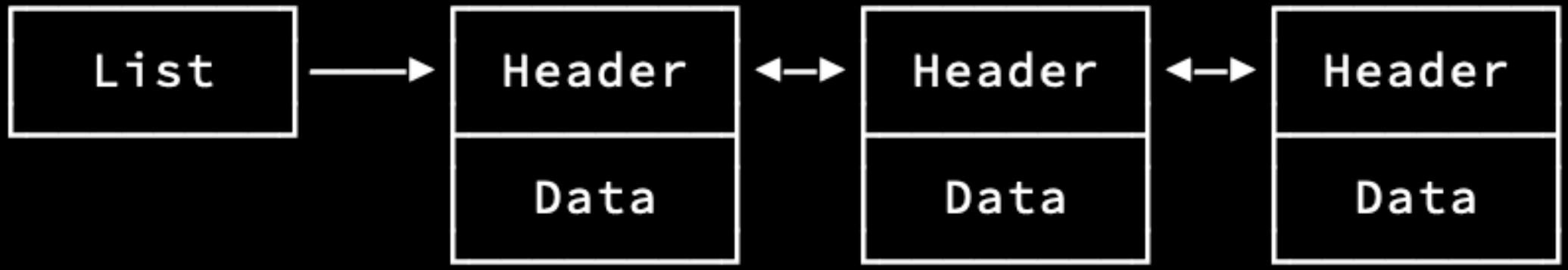
we touched on this in "what are you syncing about"

concept reminder

"classic" linked lists

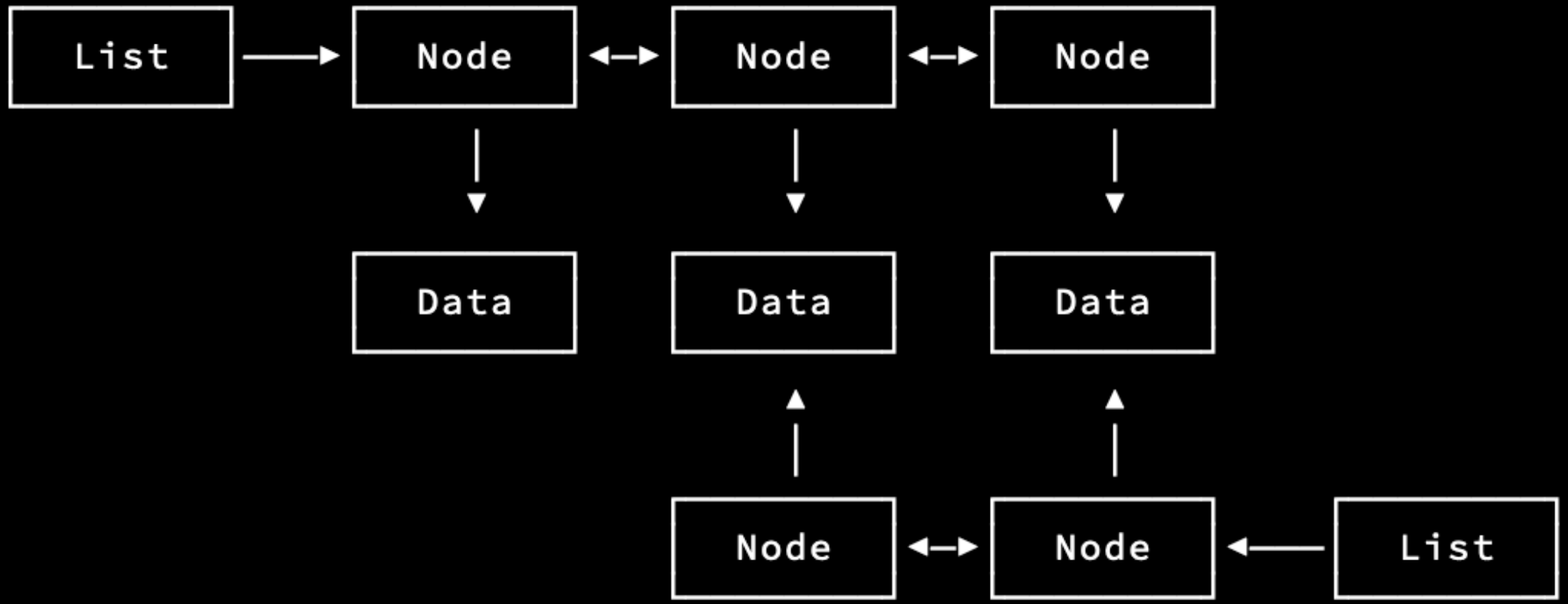

List
Node
Node
Node
Data
Data
Data

"intrusive" linked lists


List
Header
Data
Header
Data
Header
Data

“classic” down side: two allocations for every node

"classic" up side: items can exist in multiple lists


List
Node
Node
Node
Data
Data
Data
Node
Node
List

but linked lists are bad, right?

hah hah! we're doing embedded systems!

can't have a cache miss if you don't have cache

can't have a vec if you don't have alloc

fun fact: this is listed as an "ugh, whatever" note on "Too Many Linked Lists"

Mumble mumble kernel embedded something something intrusive.

It's niche. You're talking about a situation where you're not even using your language's *runtime*. Is that not a red flag that you're doing something strange?

It's also wildly unsafe.

But sure. Build your awesome zero-allocation lists on the stack.

so how do you have a "variable" quantity w/o vec?

usually:
upper-bound
collections

down side:
potential waste

down side: generics

maybe linked lists aren't so bad?

but how?

maitake-sync

WaitQueue vs WaitCell

but how?

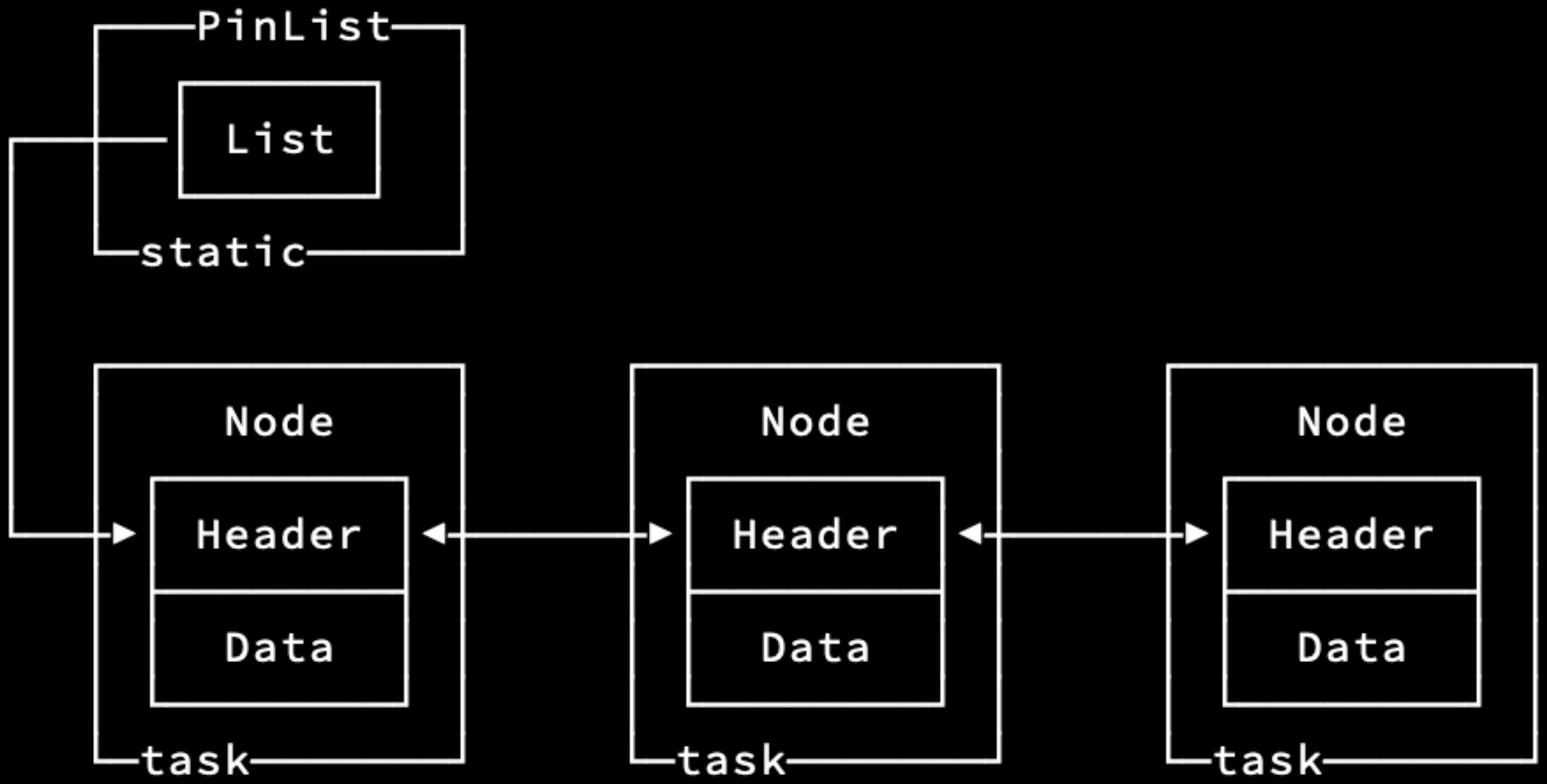
Wakers are pinned inside their tasks

when a Waker is dropped, we remove it from the list

when the WaitQueue is dropped:
all items are removed from the list

Pin guarantees we can't skip the drop

I made a crate for this: PinList


PinList
List
static
Node
Header
Data
task
Node
Header
Data
task
Node
Header
Data
task

tasks can "lend" their ephemeral data to the list

but wait, how do we put non-'static data in a static list?

it's unsafe, but it's actually safe.

you can only access list items while holding a mutex

running drop means holding the mutex

and since as long as it's on the list, it exists:

good enough!

this lets you do very cool things, like separating i/o workers from "data"

cfg-noodle: embedded storage library

ergot: embedded networking library

it's a pretty handy pattern :)