

# Hands-on: Demo Spatial (GeMM)

**Matthew Feldman**, Luigi Nardi, Artur Souza, Kunle Olukotun

# HyperMapper + Spatial Tutorial

- This GeMM app is ready for you to experiment with!
  - <https://github.com/luinardi/hypermapper>

luinardi / hypermapper

Code Issues Pull requests Actions Wiki Security Insights

Join GitHub today

GitHub is home to over 50 million developers working together to host and review code, manage projects, and build software together.

Dismiss

Sign up

Black-box Optimizer based on Bayesian Optimization

7 commits 1 branch 0 packages 0 releases 2 contributors MIT

Branch: master New pull request Find file Clone or download

luinardi Update README.md ✓ Latest commit 2beed4 8 hours ago

example\_outputs HyperMapper Release. 6 months ago

example\_scenarios Updated demo with more examples. 5 months ago

scripts Fixed issue with unordered configurations. 20 days ago

tests HyperMapper Release. 6 months ago

.gitignore Removed .idea and added .gitignore. Fixes #1 27 days ago

.travis.yml HyperMapper Release. 6 months ago

LICENSE HyperMapper Release. 6 months ago

README.md Update README.md 8 hours ago

Examples
Branin
Parameter Types
Branin4
CurrinExp
Multiple Objectives
Multi-objective Branin
Ordinal Multi-objective Branin
DTLZ1
Constrained Spaces
Chakong-Haines
Focus on Portion of the Pareto
Bounding the Multi-objective Branin Function
Client-Server Mode
Client-Server Branin
Client-Server Chakong-Haines
Use Cases
The Spatial Use Case
The SEALBenchmark Use Case
Other

# GeMM in Hardware

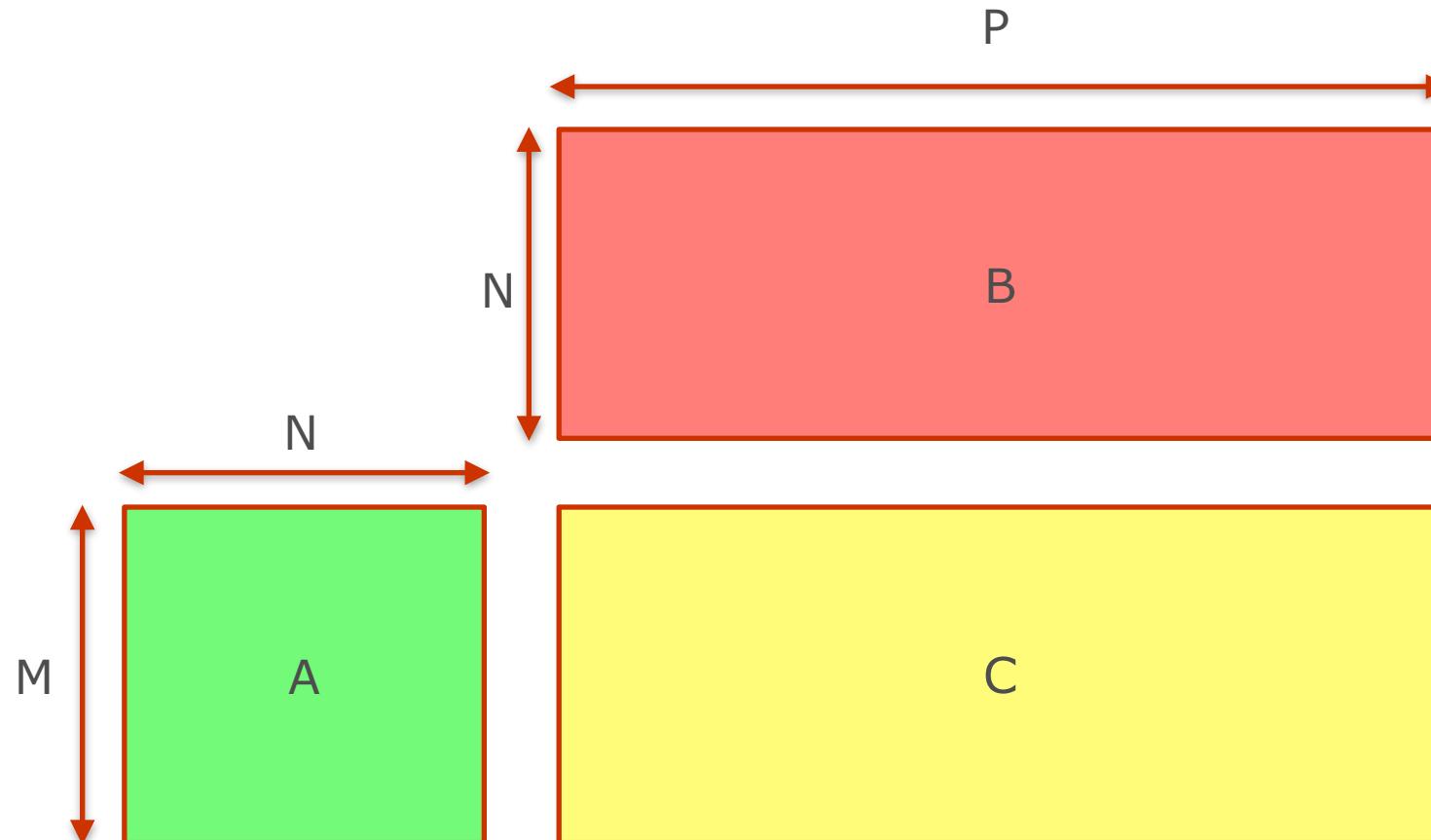
---

- **Task:** Compute  $A_{M \times N} * B_{N \times P} = C_{M \times P}$
- Let's solve this using *tiling* and *inner products*

# GeMM in Hardware

---

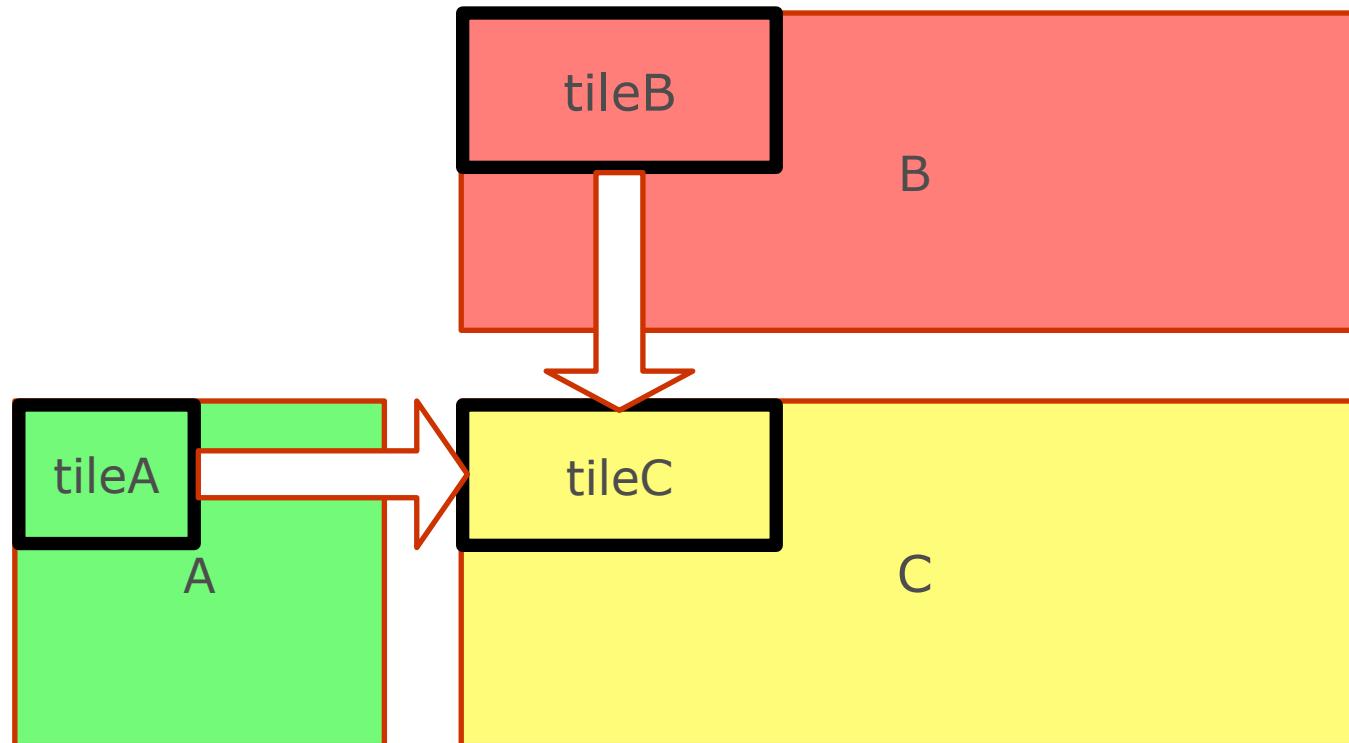
- Task: Compute  $A_{M \times N} * B_{N \times P} = C_{M \times P}$
- Let's solve this using *tiling* and *inner products*



# GeMM in Hardware

---

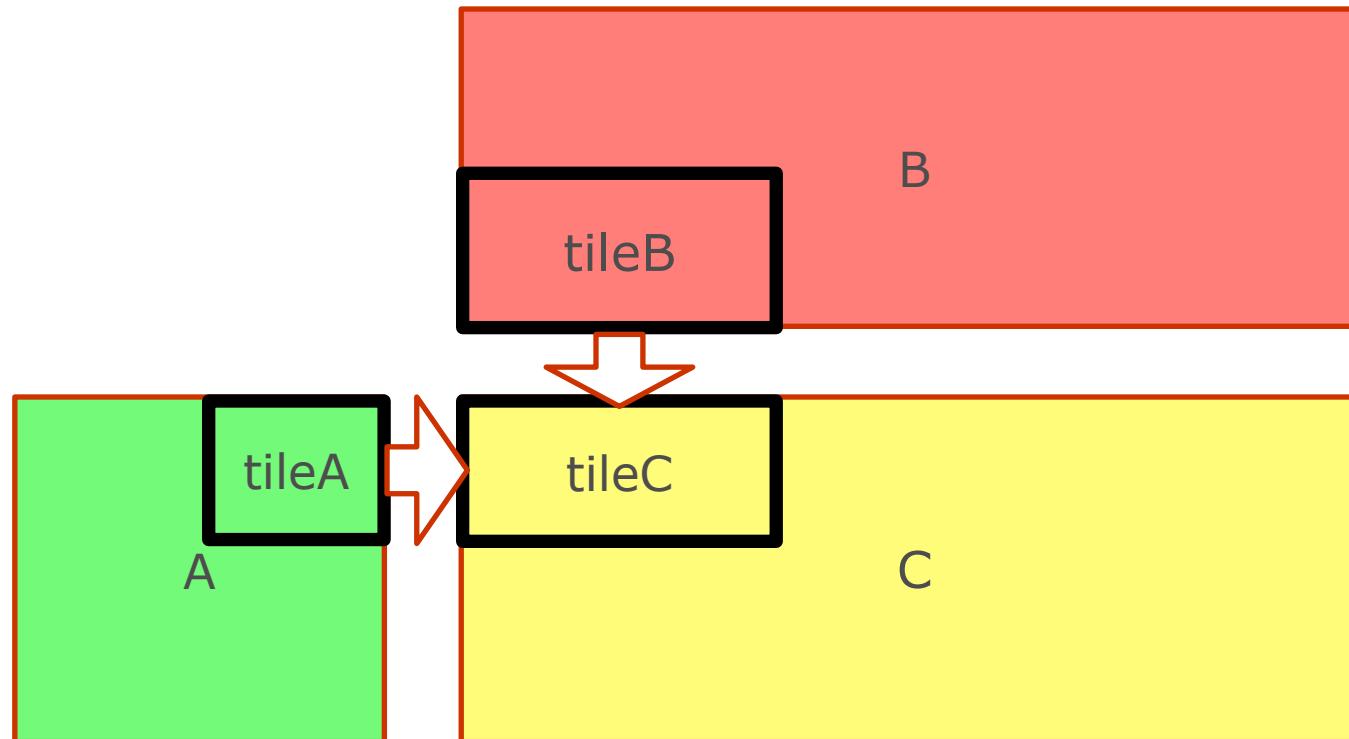
- Task: Compute  $A_{M \times N} * B_{N \times P} = C_{M \times P}$
- Let's solve this using *tiling* and *inner products*



# GeMM in Hardware

---

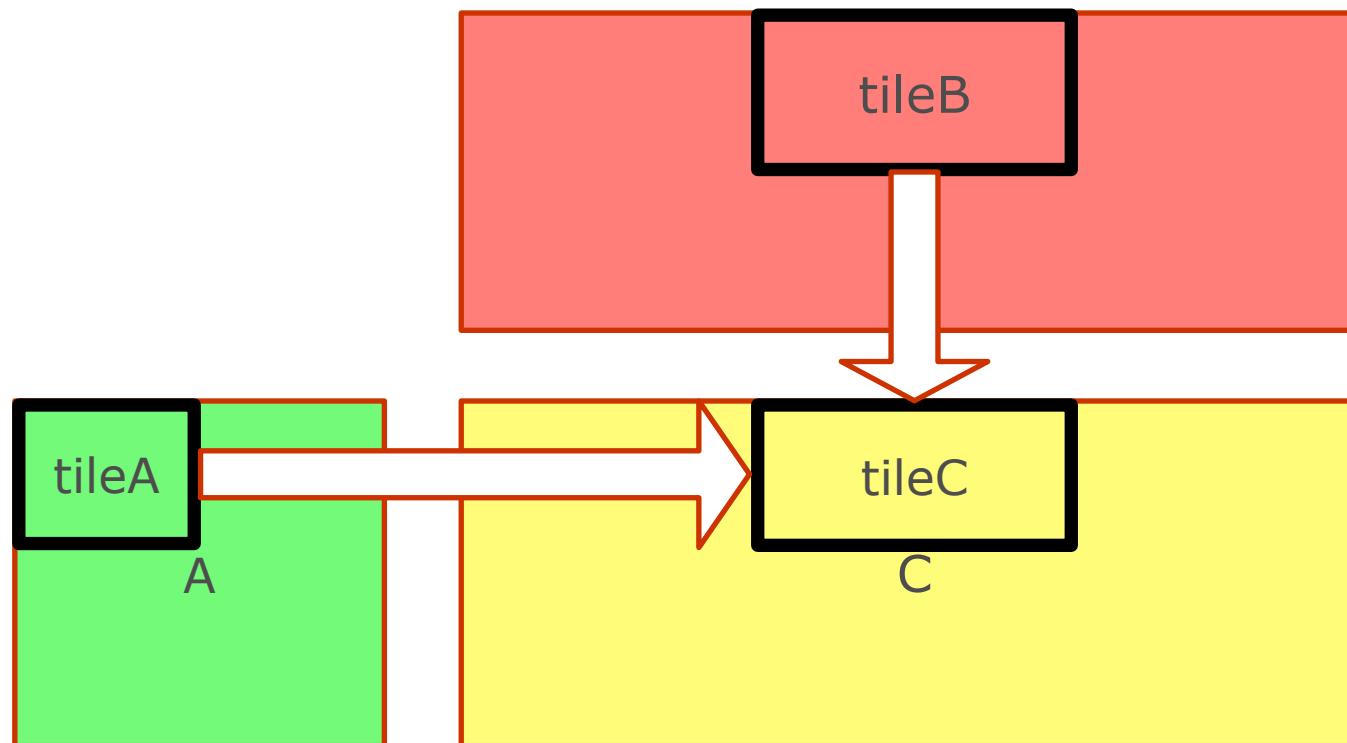
- Task: Compute  $A_{M \times N} * B_{N \times P} = C_{M \times P}$
- Let's solve this using *tiling* and *inner products*



# GeMM in Hardware

---

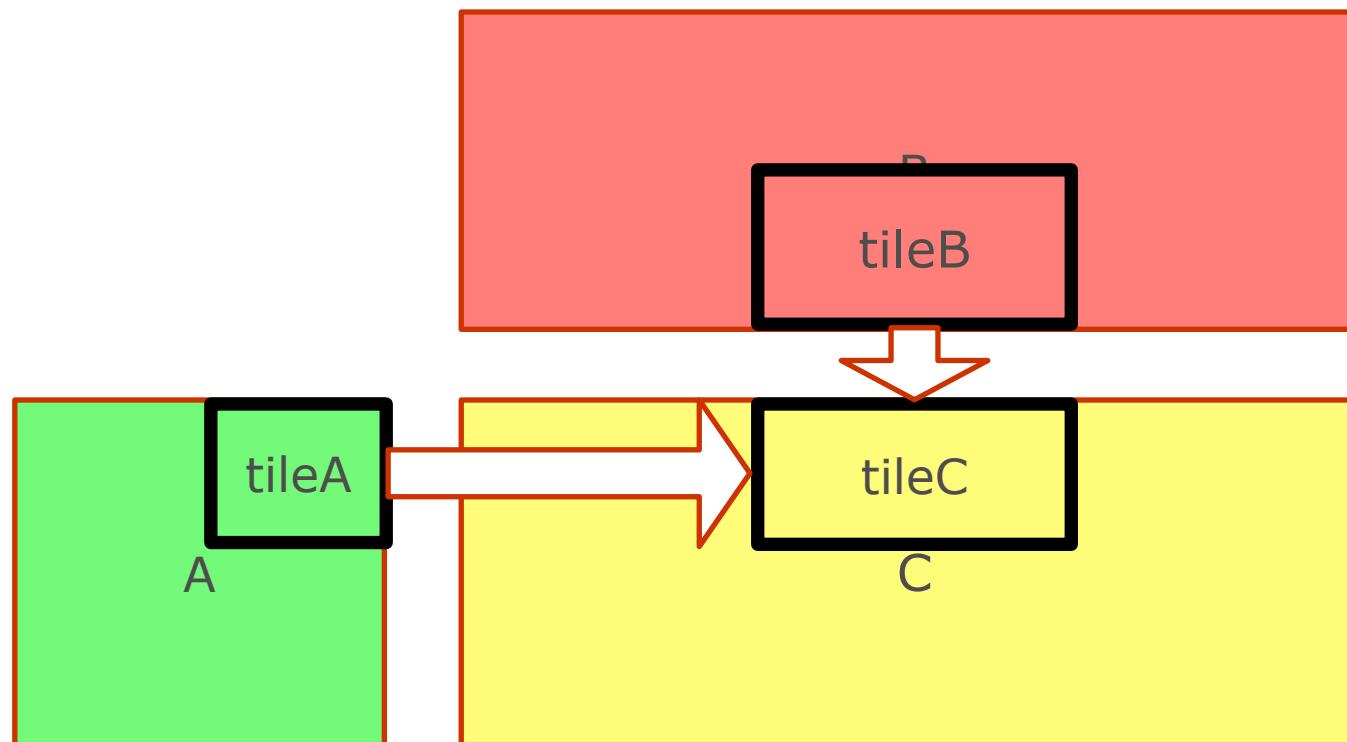
- Task: Compute  $A_{M \times N} * B_{N \times P} = C_{M \times P}$
- Let's solve this using *tiling* and *inner products*



# GeMM in Hardware

---

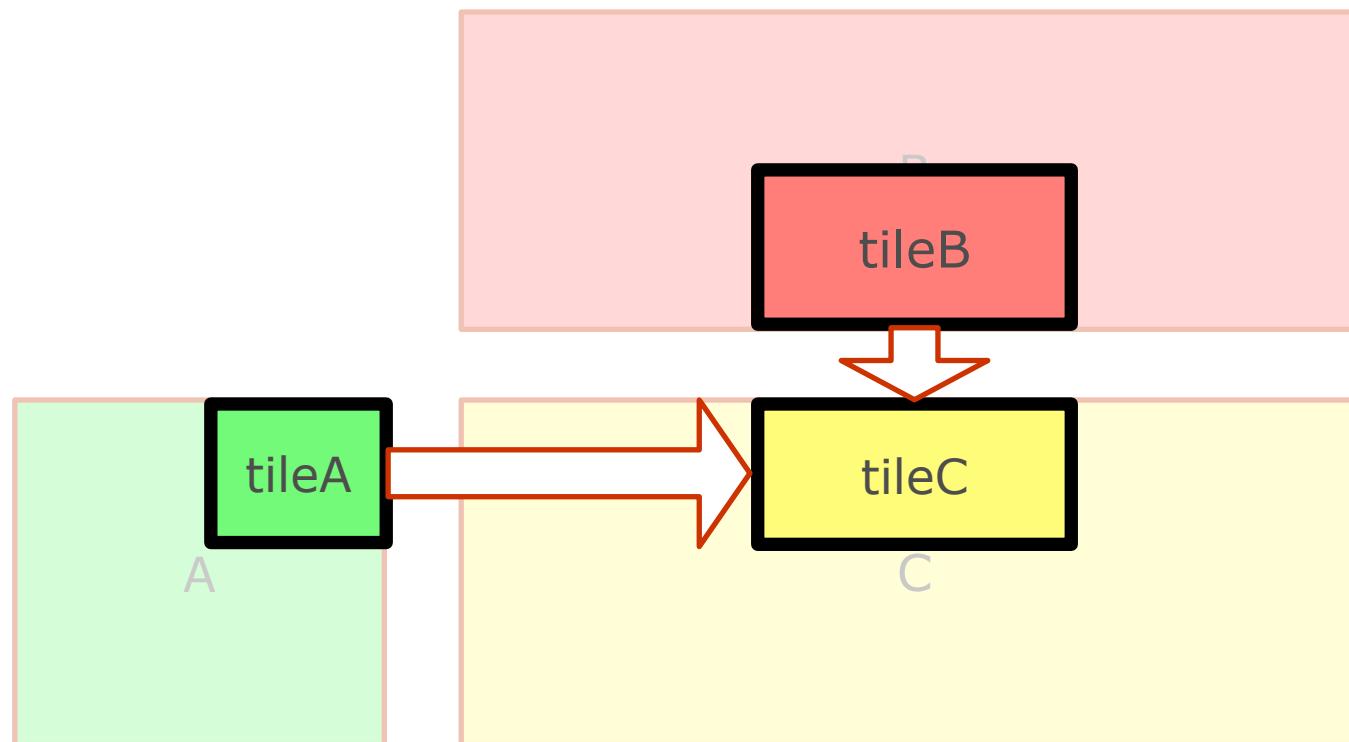
- Task: Compute  $A_{M \times N} * B_{N \times P} = C_{M \times P}$
- Let's solve this using *tiling* and *inner products*



# GeMM in Hardware

---

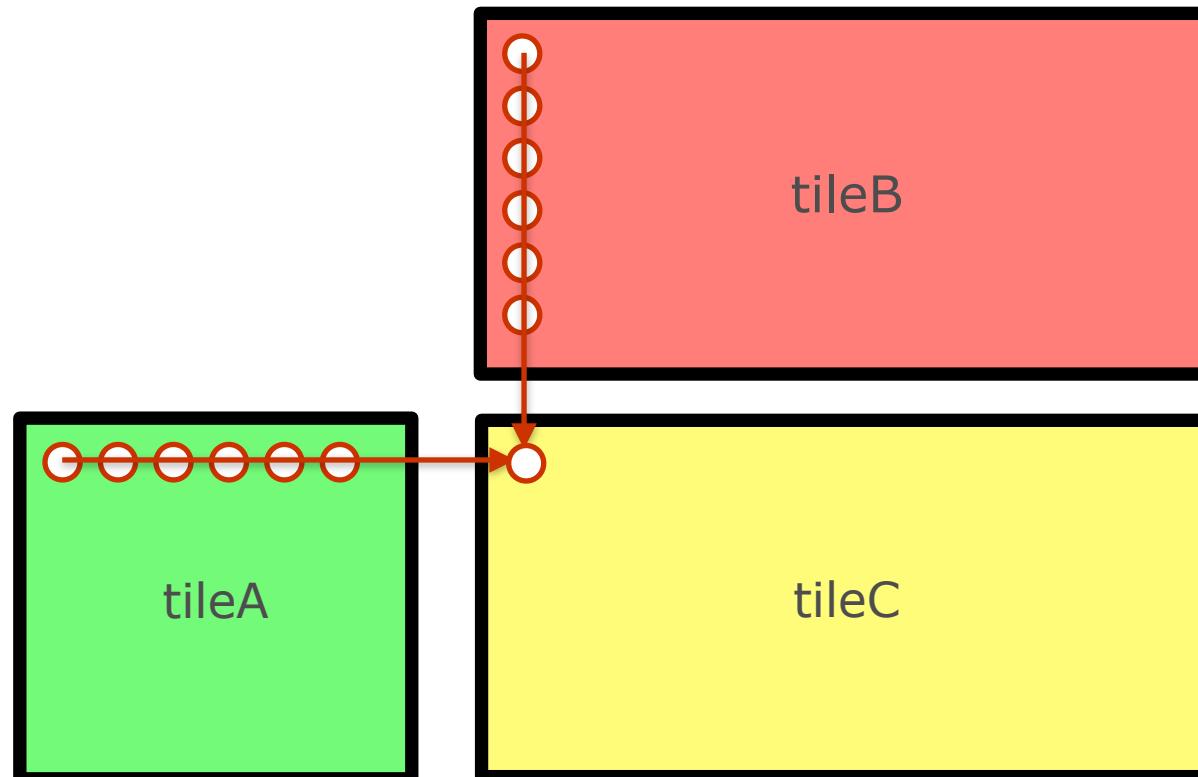
- Task: Compute  $A_{M \times N} * B_{N \times P} = C_{M \times P}$
- Let's solve this using *tiling* and *inner products*



# GeMM in Hardware

---

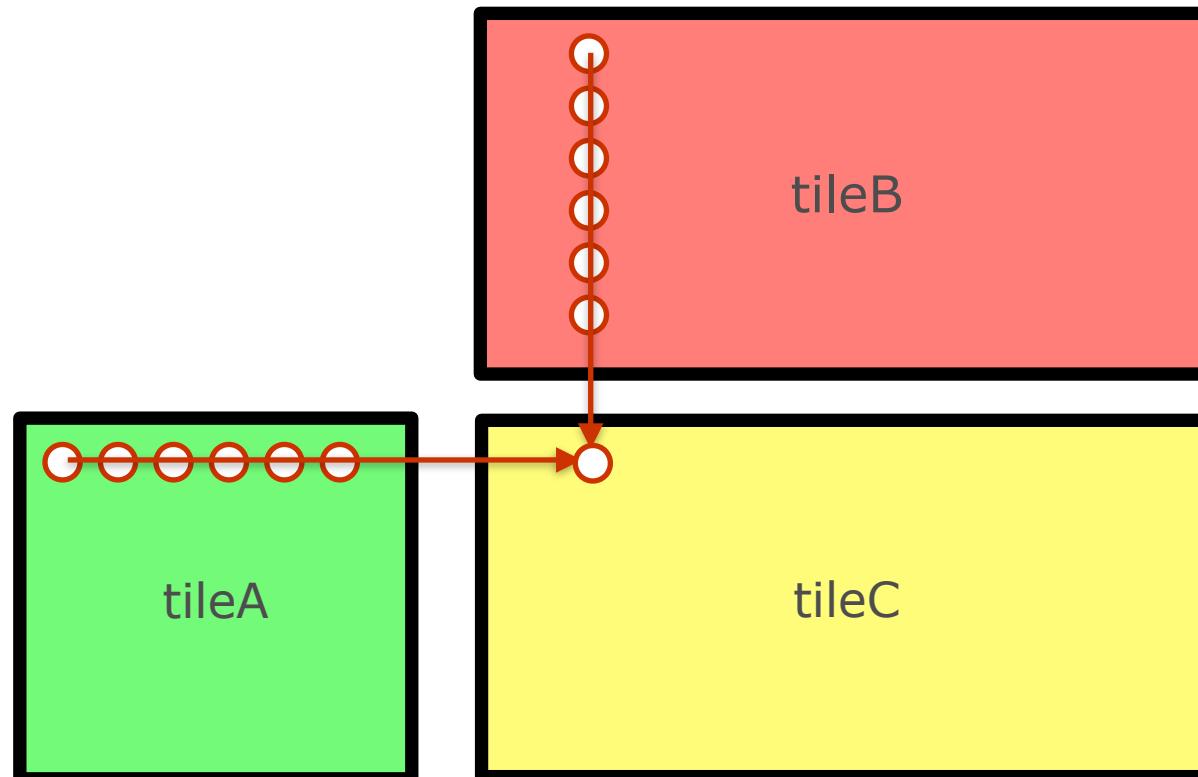
- Task: Compute  $A_{M \times N} * B_{N \times P} = C_{M \times P}$
- Let's solve this using *tiling* and *inner products*



# GeMM in Hardware

---

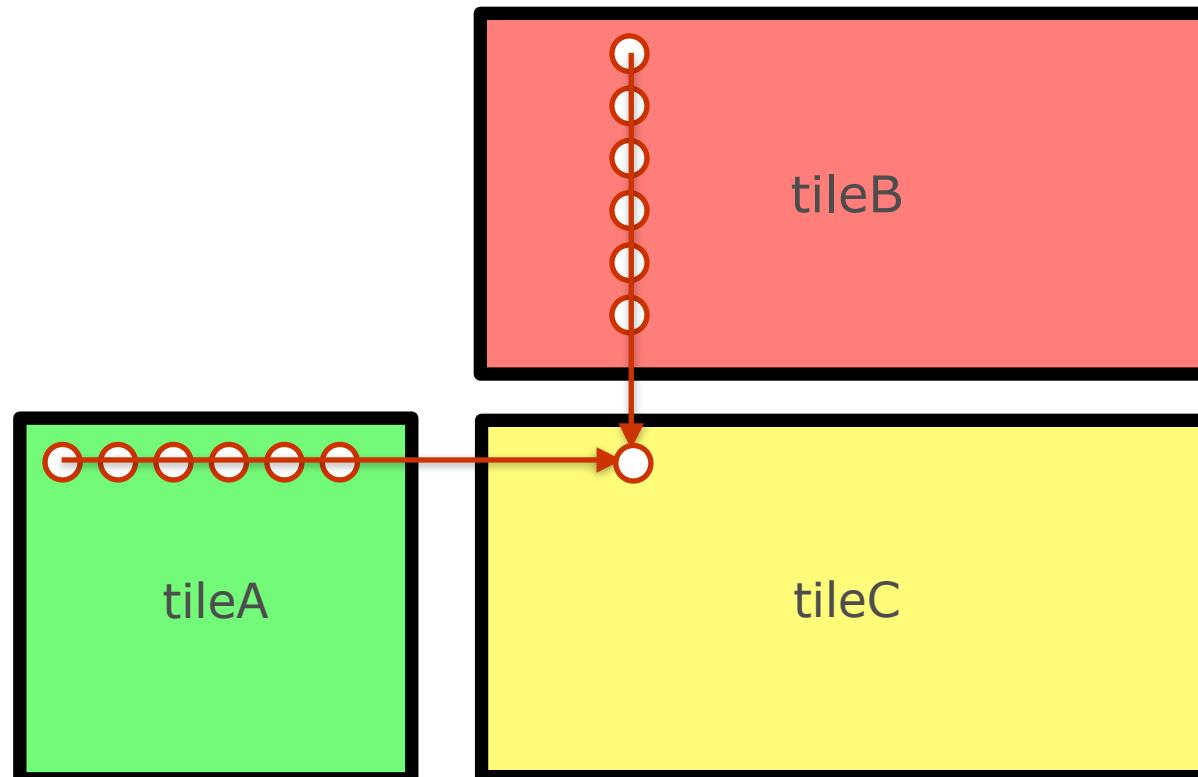
- Task: Compute  $A_{M \times N} * B_{N \times P} = C_{M \times P}$
- Let's solve this using *tiling* and *inner products*



# GeMM in Hardware

---

- Task: Compute  $A_{M \times N} * B_{N \times P} = C_{M \times P}$
- Let's solve this using *tiling* and *inner products*



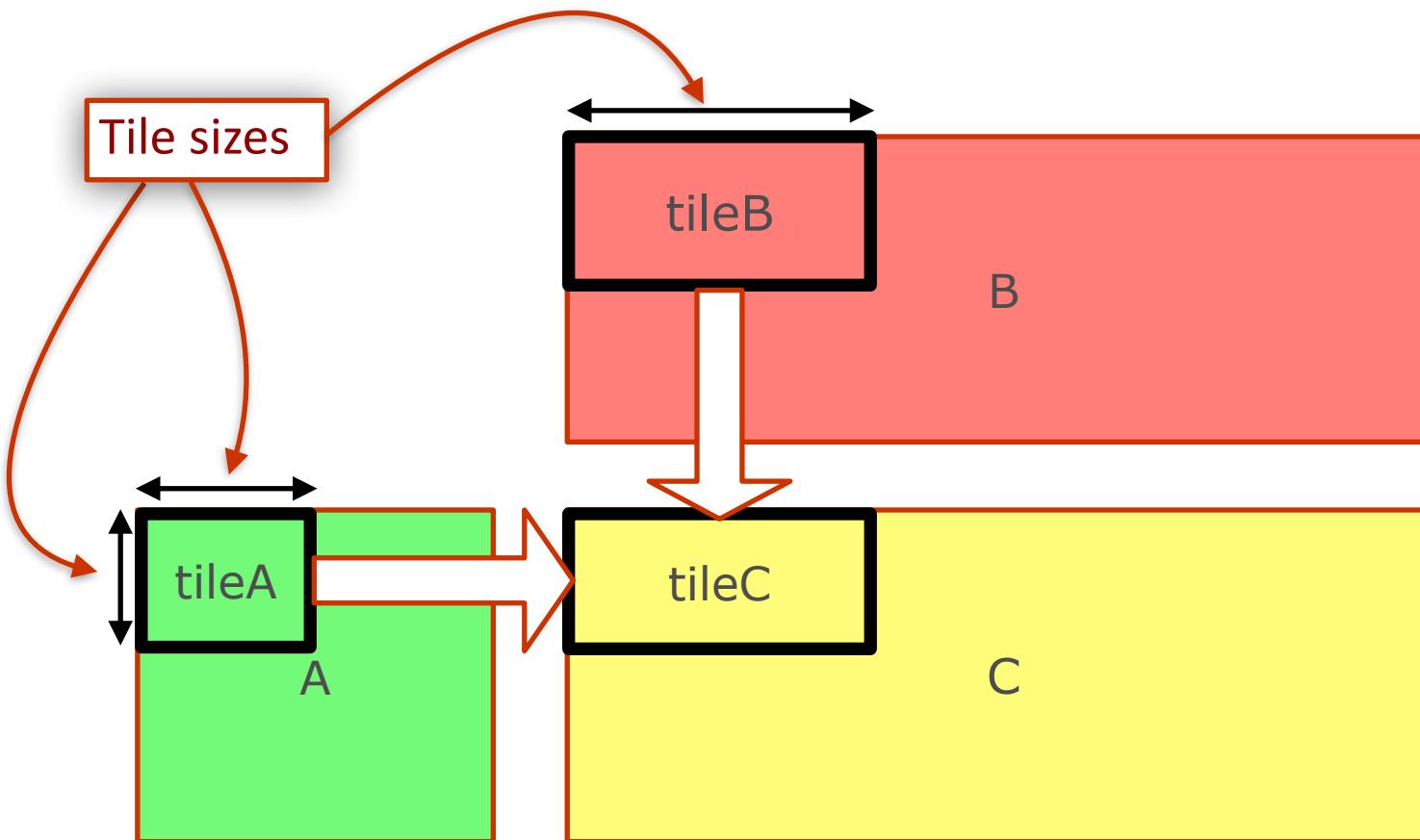
# GeMM in Hardware

---

- There are a few obvious design choices...

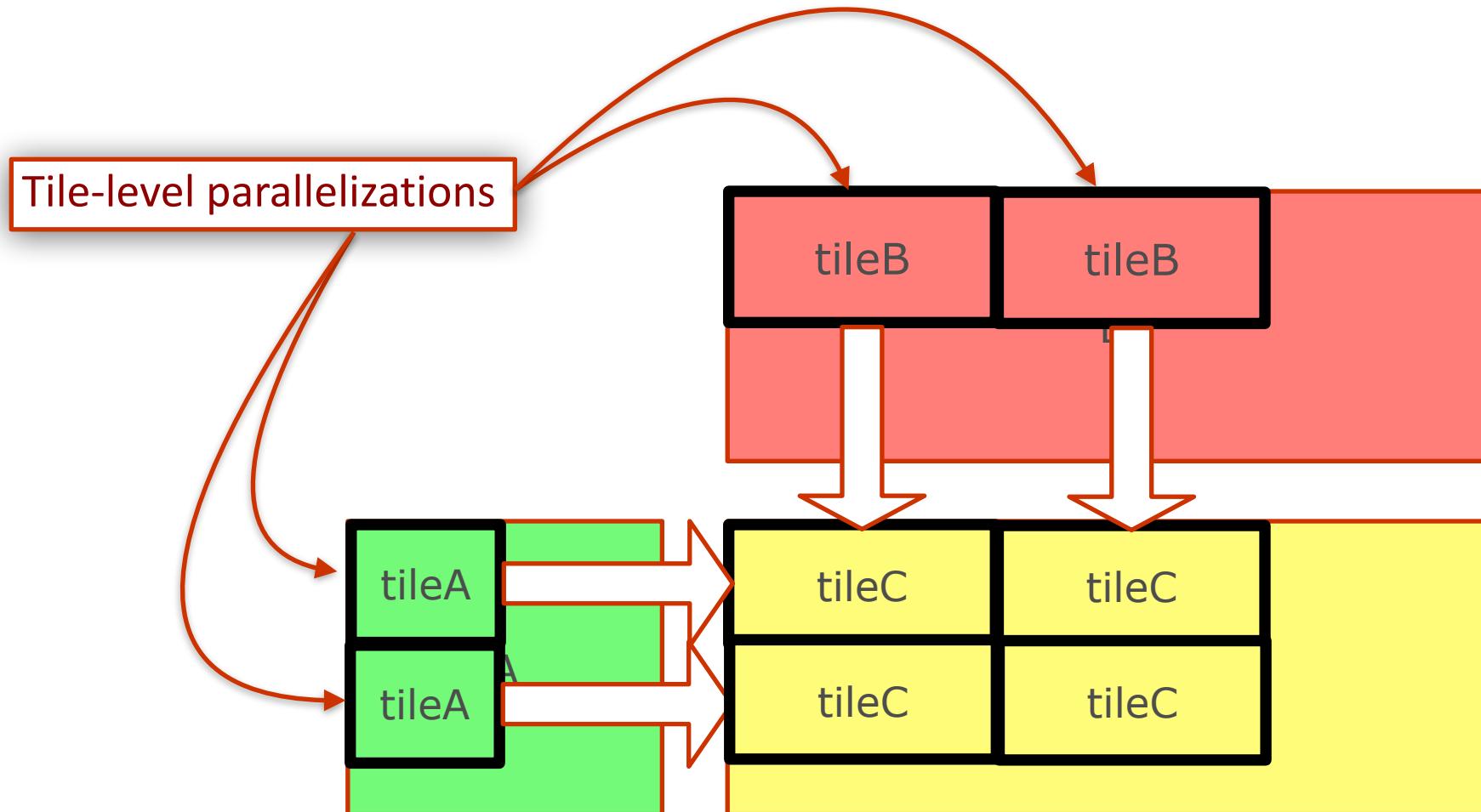
# GeMM in Hardware

- There are a few obvious design choices...



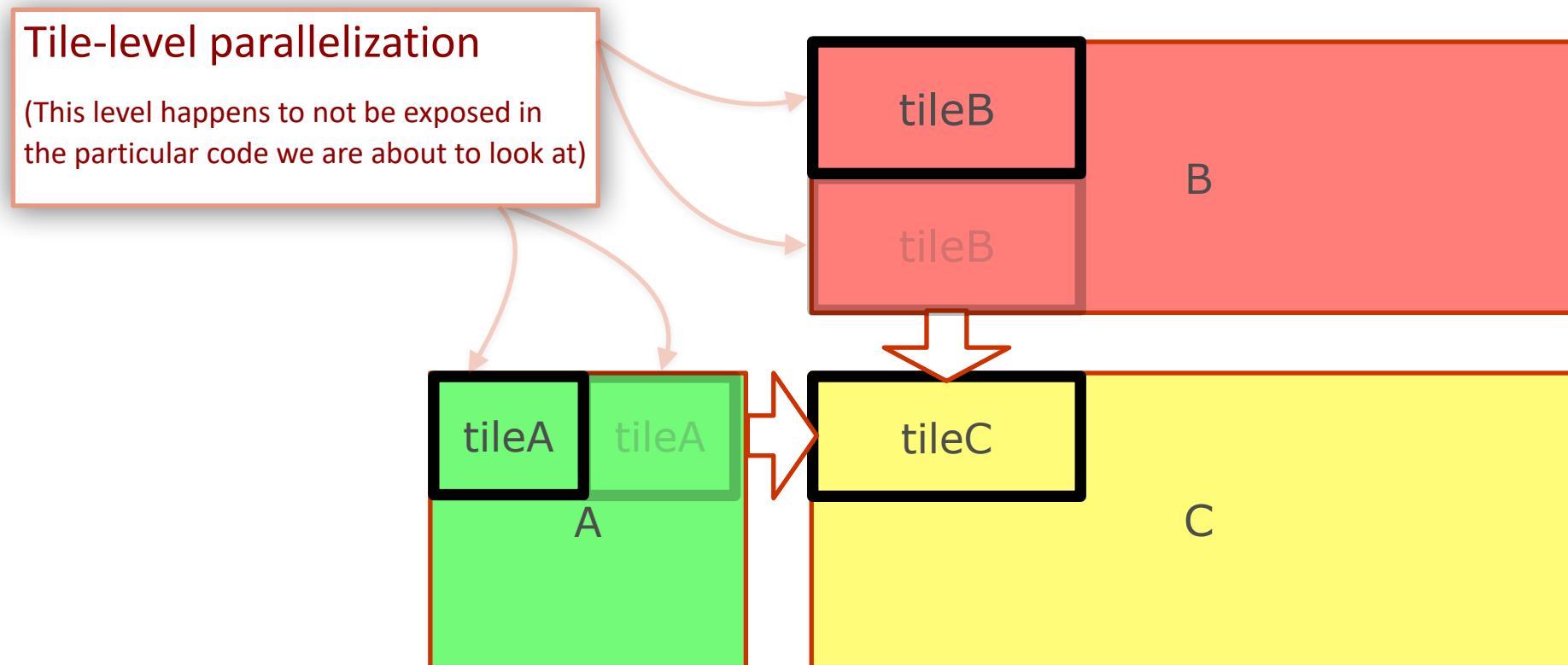
# GeMM in Hardware

- There are a few obvious design choices...



# GeMM in Hardware

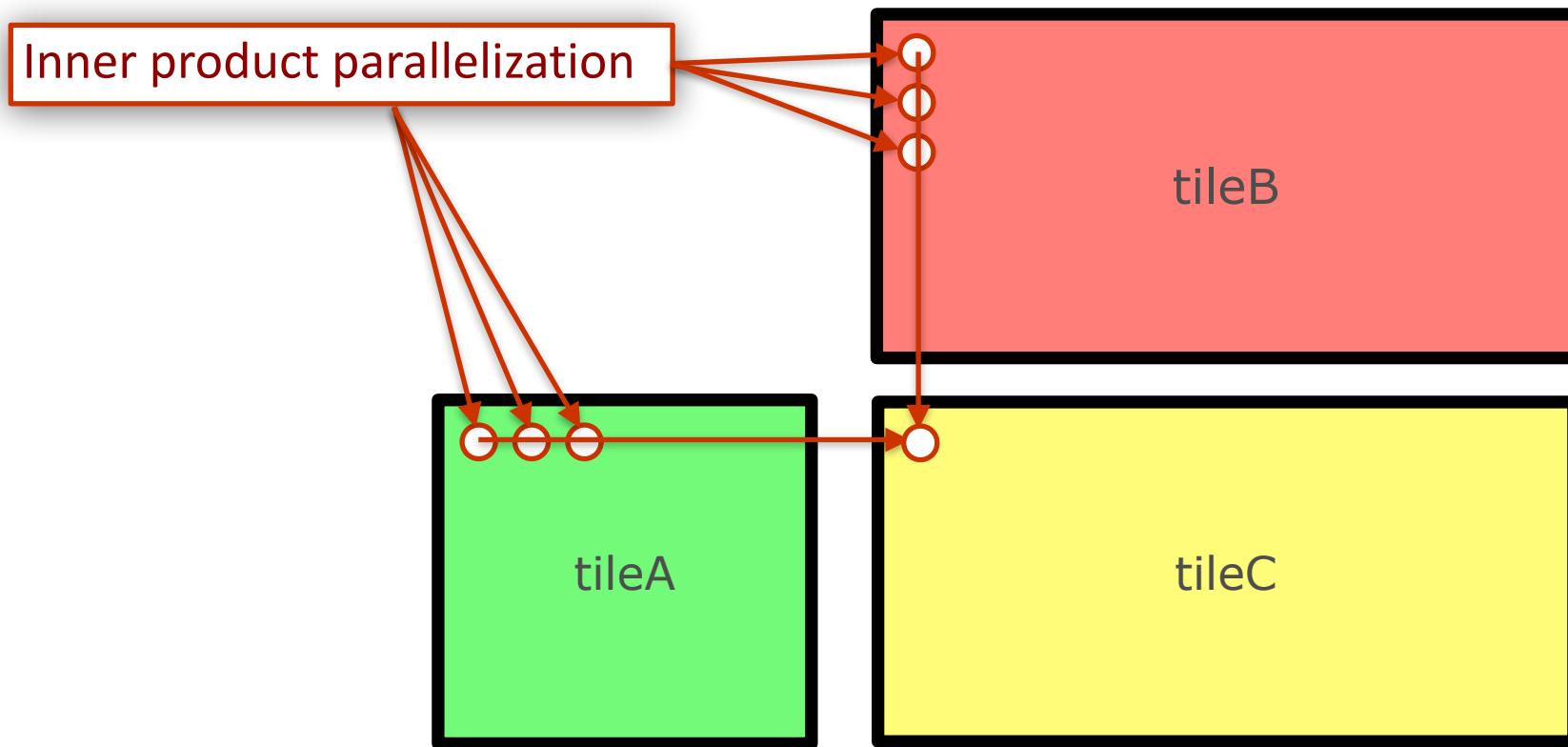
- There are a few obvious design choices...



# GeMM in Hardware

---

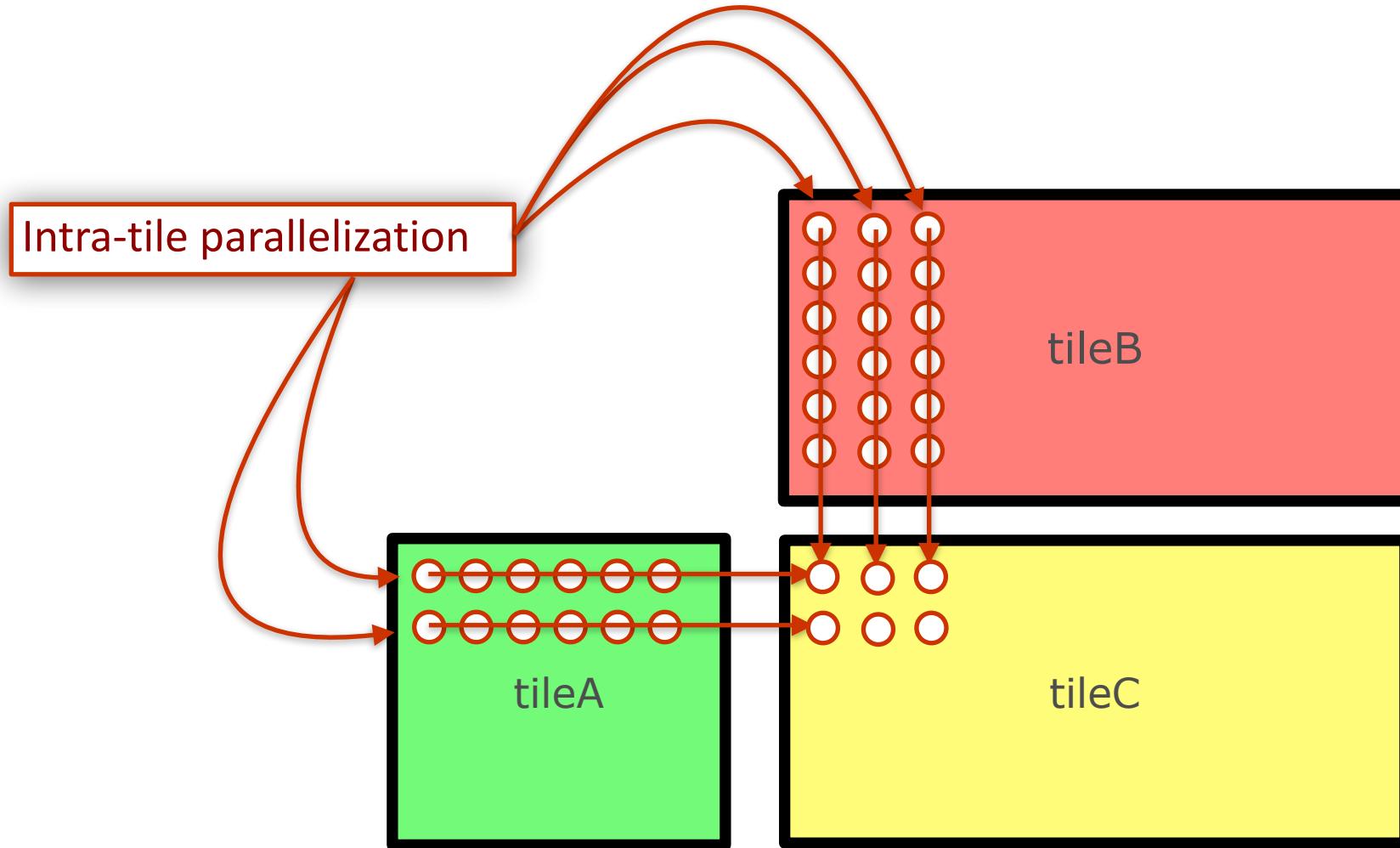
- There are a few obvious design choices...



# GeMM in Hardware

---

- There are a few obvious design choices...



# GeMM in Hardware

---

- There are a few obvious design choices...
  - Tile sizes
  - Tile-level parallelization
  - Intra-tile parallelization
  - Inner product parallelization

# GeMM in Hardware

---

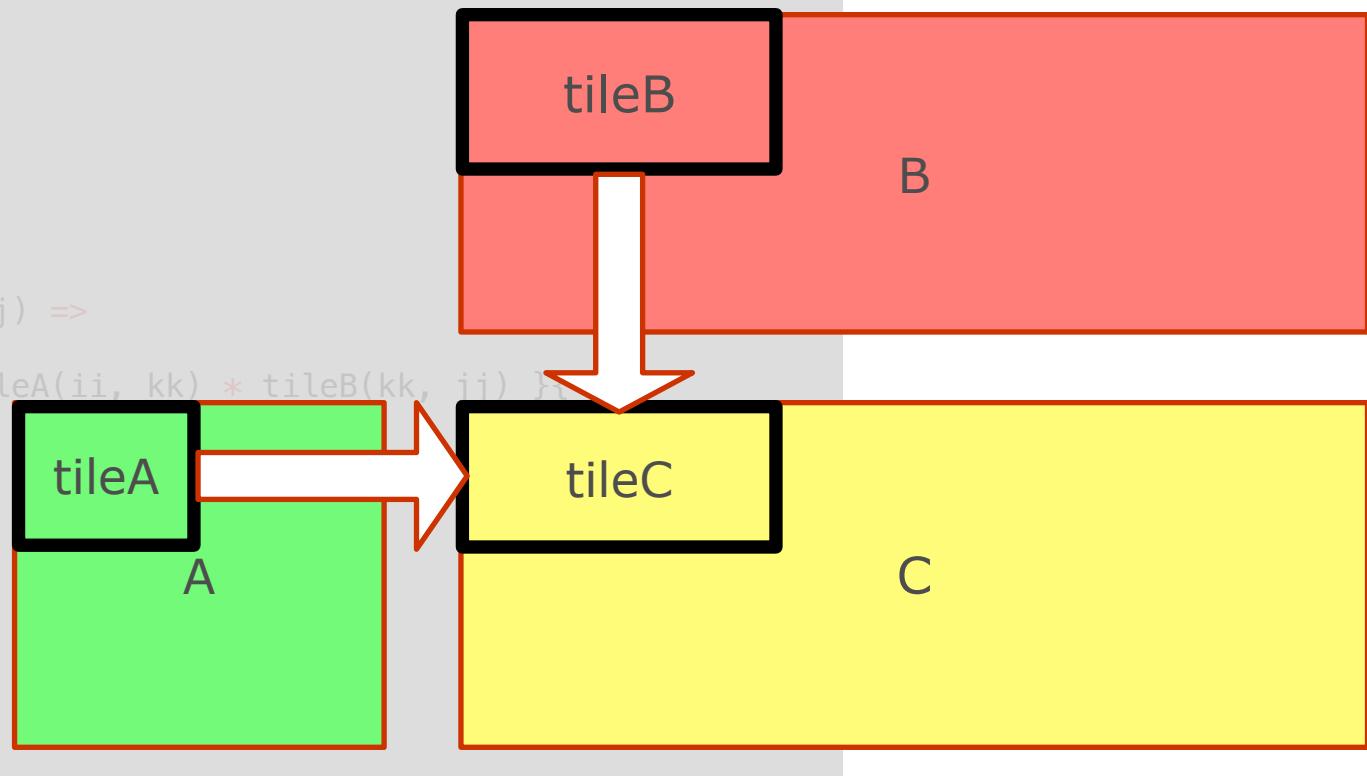
- There are a few obvious design choices...
  - Tile sizes
  - Tile-level parallelization
  - Intra-tile parallelization
  - Inner product parallelization
- ... And some less obvious choices
  - Outer controller scheduling (Sequential or Pipeline)
  - DRAM transfer parallelization

# GeMM in Hardware - Spatial App

```
Foreach(M by bm par tileM_par, P by bp par tileP_par){(i,j) =>
    val tileC = SRAM[T](bm, bp)
    Foreach(N by bn){k =>
        val tileA = SRAM[T](bm, bn)
        val tileB = SRAM[T](bn, bp)
        tileA load A(i::i+bm, k::k+bn par load_par)
        tileB load B(k::k+bn, j::j+bp par load_par)
        Foreach(bm by 1 par M_par, bp by 1 par P_par){ (ii,jj) =>
            val prod = Reduce(Reg[T])(bn by 1 par ip){kk => tileA(ii, kk) * tileB(kk, jj) }{+_}
            val prev = mux(k == 0, 0.to[T], tileC(ii,jj))
            tileC(ii,jj) = prev + prod.value
        }
    }
    C(i::i+bm, j::j+bp par store_par) store tileC
}
```

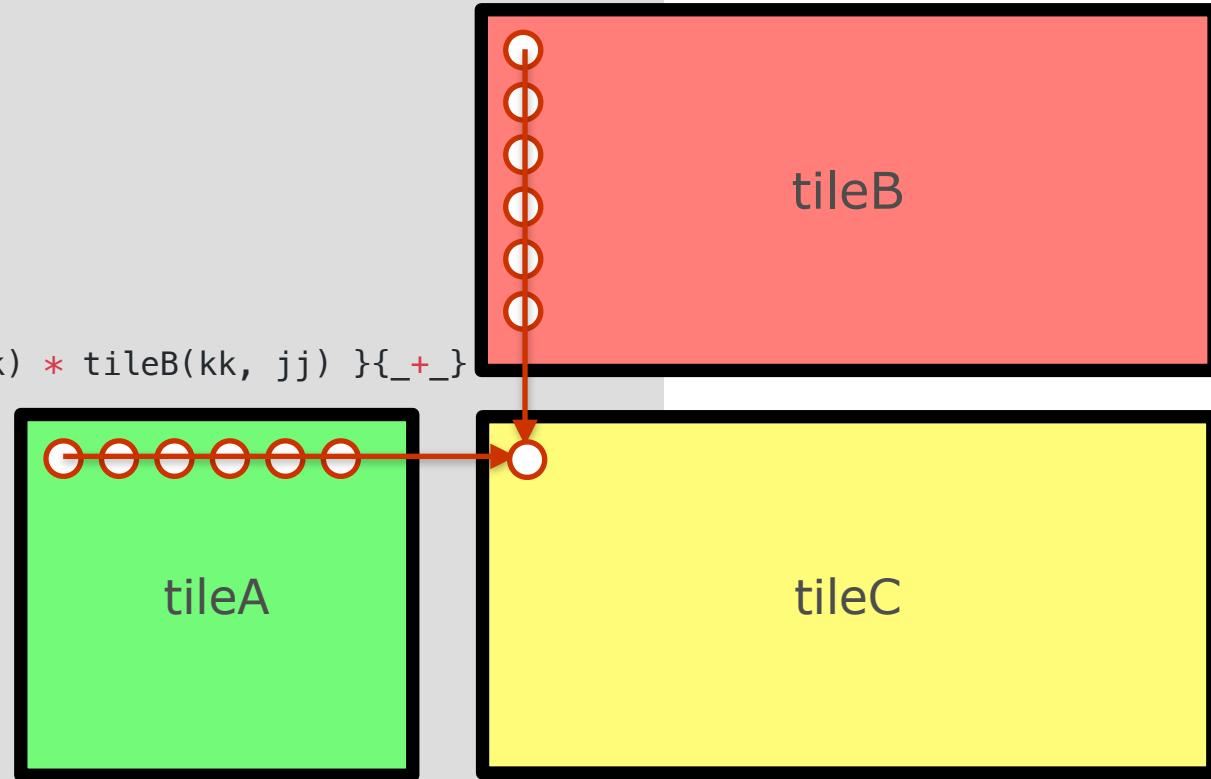
# GeMM in Hardware - Spatial App

```
Foreach(M by bm par tileM_par, P by bp par tileP_par){(i,j) =>
  val tileC = SRAM[T](bm, bp)
  Foreach(N by bn){k =>
    val tileA = SRAM[T](bm, bn)
    val tileB = SRAM[T](bn, bp)
    tileA load A(i::i+bm, k::k+bn par load_par)
    tileB load B(k::k+bn, j::j+bp par load_par)
    Foreach(bm by 1 par M_par, bp by 1 par P_par){ (ii,jj) =>
      val prod = Reduce(Reg[T])(bn by 1 par ip){kk => tileA(ii, kk) * tileB(kk, jj)}
      val prev = mux(k == 0, 0.to[T], tileC(ii,jj))
      tileC(ii,jj) = prev + prod.value
    }
    C(i::i+bm, j::j+bp par store_par) store tileC
  }
}
```



# GeMM in Hardware - Spatial App

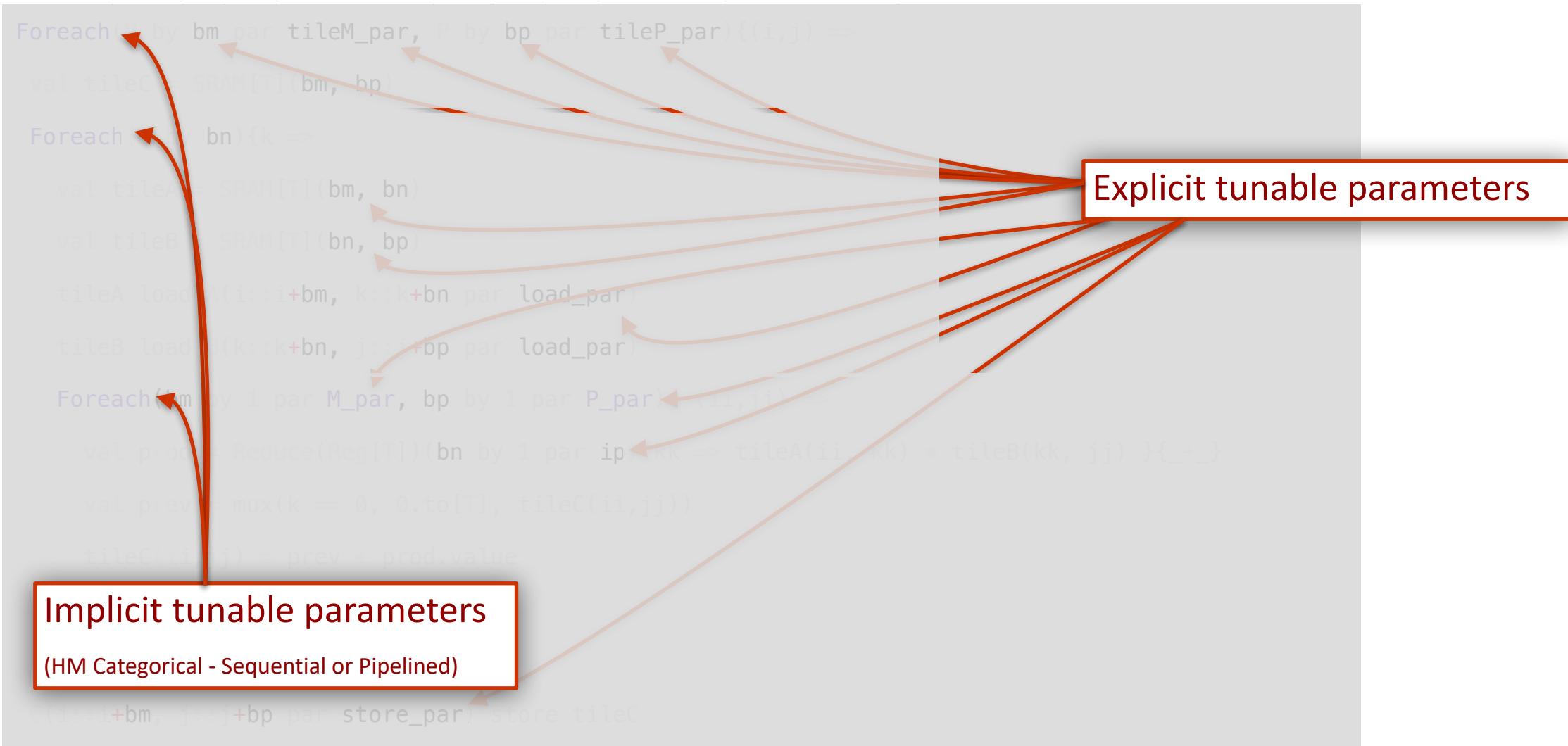
```
Foreach(M by bm par tileM_par, P by bp par tileP_par){(i,j) =>
  val tileC = SRAM[T](bm, bp)
  Foreach(N by bn){k =>
    val tileA = SRAM[T](bm, bn)
    val tileB = SRAM[T](bn, bp)
    tileA load A(i::i+bm, k::k+bn par load_par)
    tileB load B(k::k+bn, j::j+bp par load_par)
    Foreach(bm by 1 par M_par, bp by 1 par P_par){ (ii,jj) =>
      val prod = Reduce(Reg[T])(bn by 1 par ip){kk => tileA(ii, kk) * tileB(kk, jj) }{+_}
      val prev = mux(k == 0, 0.to[T], tileC(ii,jj))
      tileC(ii,jj) = prev + prod.value
    }
  C(i::i+bm, j::j+bp par store_par) store tileC}
```



# GeMM in Hardware - Spatial App

```
Foreach(M by bm par tileM_par, P by bp par tileP_par){(i,j) =>
  val tileC = SRAM[T](bm, bp)
  Foreach(N by bn){k =>
    val tileA = SRAM[T](bm, bn)
    val tileB = SRAM[T](bn, bp)
    tileA load A(i::i+bm, k::k+bn par load_par)
    tileB load B(k::k+bn, j::j+bp par load_par)
    Foreach(bm by 1 par M_par, bp by 1 par P_par){ (ii,jj) =>
      val prod = Reduce(Reg[T])(bn by 1 par ip){kk => tileA(ii, kk) * tileB(kk, jj) }{+_}
      val prev = mux(k == 0, 0.to[T], tileC(ii,jj))
      tileC(ii,jj) = prev + prod.value
    }
  }
  C(i::i+bm, j::j+bp par store_par) store tileC
}
```

# GeMM in Hardware - Spatial App



# GeMM in Hardware - Spatial App

- Parameter spaces can be specified:

```
// Expose tile sizes as DSE parameters

val bm = 16 (1 -> 128)
val bn = 16 (16 -> 16 -> 1028)
val bp = 16 (16 -> 16 -> 1028)

// Expose parallelization factors as DSE parameters

val tileM_par = 2 (1 -> 6)
val tileP_par = 2 (1 -> 6)
val M_par = 2 (1 -> 8)
val P_par = 2 (1 -> 8)
val ip = 2 (1 -> 64)
val load_par = 4 (1,2,4,8,16)
val store_par = 4 (1,2,4,8,16)
```

# GeMM in Hardware - Spatial App

- Parameter spaces can be specified:

```
// Expose tile sizes as DSE parameters
```

```
val bm = 16 (1 → 128) ←
```

```
val bn = 16 (16 → 16 → 1028) ←
```

```
val bp = 16 (16 → 16 → 1028) ←
```

```
// Expose parallelization factors as DSE parameters
```

```
val tileM_par = 2 (1 → 6) ←
```

```
val tileP_par = 2 (1 → 6) ←
```

```
val M_par = 2 (1 → 8) ←
```

```
val P_par = 2 (1 → 8) ←
```

```
val ip = 2 (1 → 64) ←
```

```
val load_par = 4 (1,2,4,8,16) ←
```

```
val store_par = 4 (1,2,4,8,16) ←
```

Start -> Stop range (HM Integer)

Start -> Step -> Stop range (HM Ordinal)

Explicit possibilities (HM Ordinal)

# GeMM in Hardware - Spatial App

---

- Default values for dynamic variables can be set:

```
// Create input arguments for sizing DRAM regions (matrices)

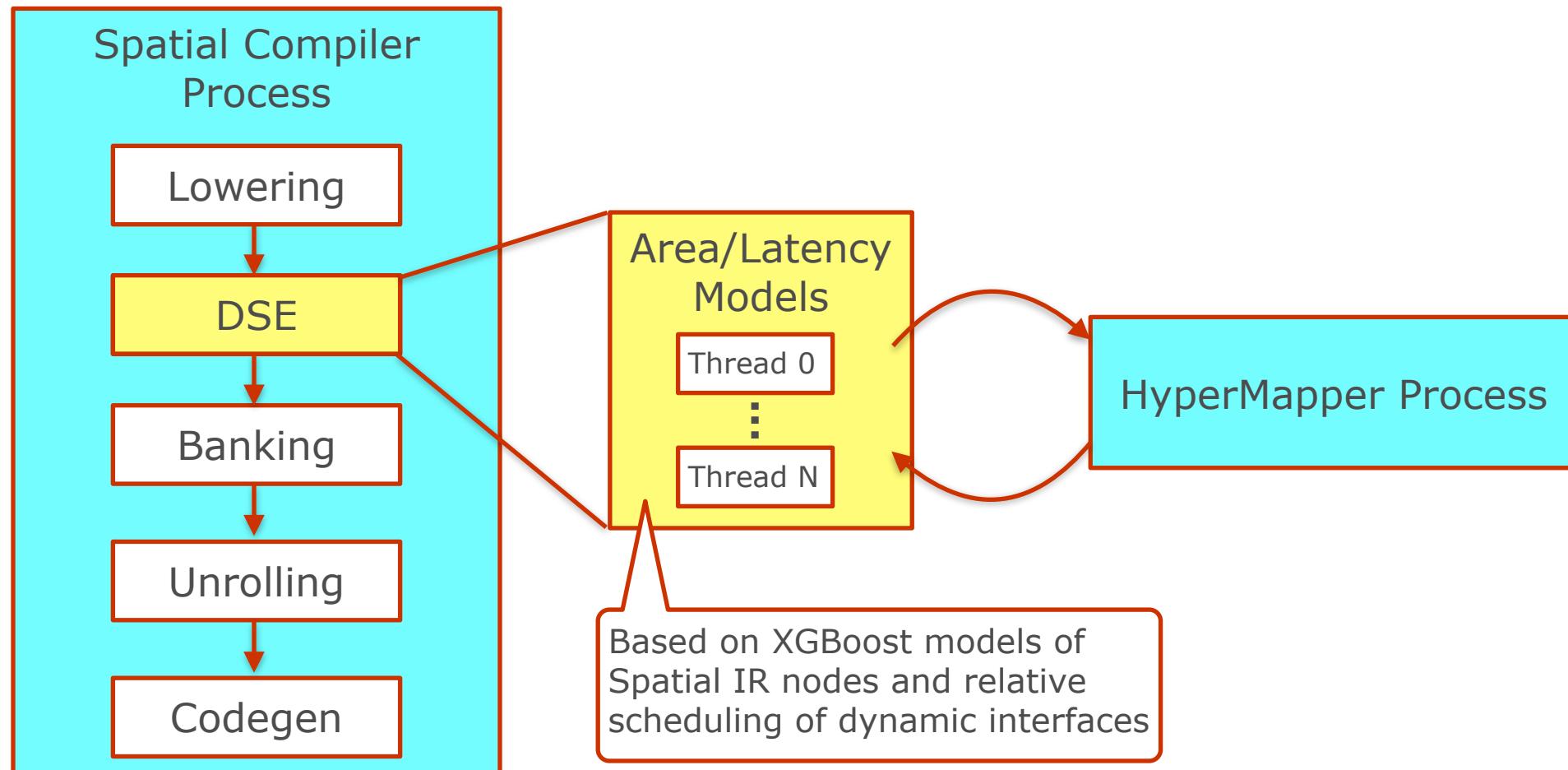
val m = args(0).toInt
val n = args(1).toInt
val p = args(2).toInt

// Set default values for runtime args

val bound(m) = 512
val bound(n) = 256
val bound(p) = 512
```

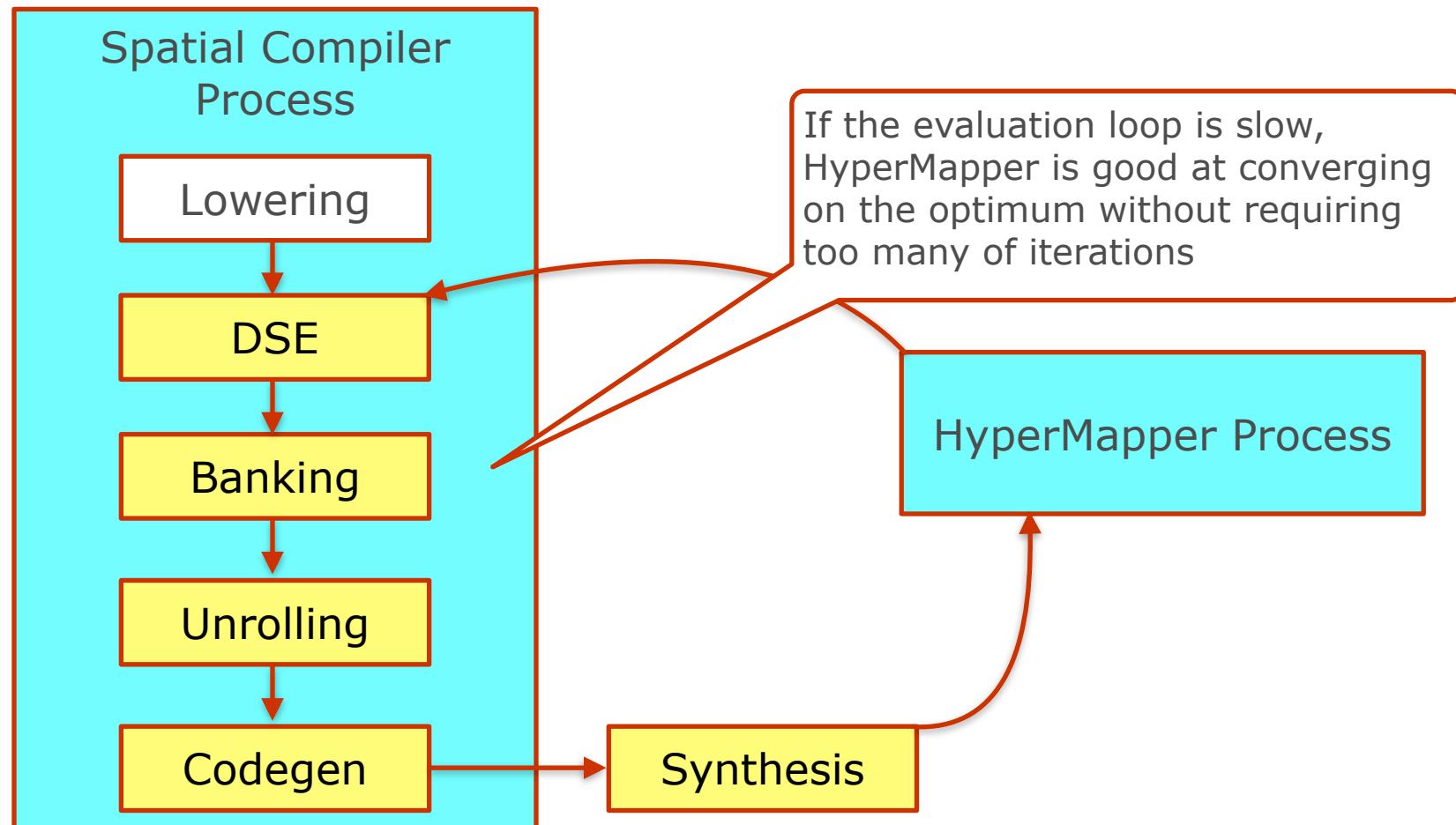
# Spatial - Hypermapper Communication

- The design space can become huge!
- HyperMapper allows us to explore it intelligently.

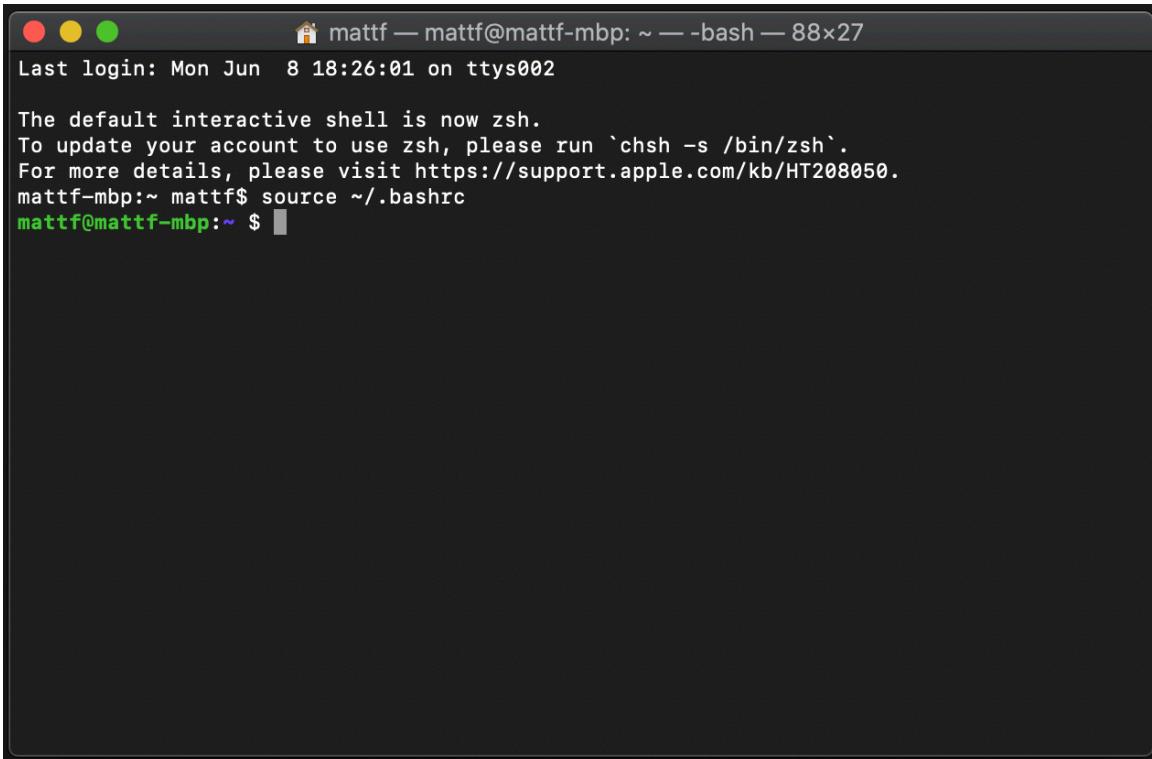


# Spatial - Hypermapper Communication

- The design space can become huge!
- HyperMapper allows us to explore it intelligently.



# Live Demo



A screenshot of a macOS terminal window. The title bar shows the user name 'mattf' and the host 'mattf@mattf-mbp'. The window title is '-bash - 88x27'. The status bar at the bottom indicates the last login was on Monday, June 8, 2021, at 18:26:01 on ttys002. The main text area displays a message about the default interactive shell being zsh, instructions to run 'chsh -s /bin/zsh' to update the account, and a link to Apple's support page for more details. The prompt 'mattf@mattf-mbp:~ \$' is visible at the bottom.

```
mattf — mattf@mattf-mbp: ~ — -bash — 88x27
Last login: Mon Jun  8 18:26:01 on ttys002
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
mattf-mbp:~ mattf$ source ~/.bashrc
mattf@mattf-mbp:~ $
```