Dynamic Insertion Policy (DIP):

→ LRU fails in thrashing workloads

→ Application types
* Cache friendly
* Scanning / Stream
* Thrashing

→ Modules of replacement policy
* Insertion
* Eviction
* Promotion

Thrashing: When the CPU tries to repeatedly access data whose size is larger than the cache size, the data gets flushed out of the cache even if the intent is to use it.

LRU Insertion Policy (LIP) ⇒ Bimodal Insertion Policy (BIP) ⇒ Dynamic Insertion Policy (DIP)

Cache replacement
- → Victim selection policy : Which line is evicted
- → Insertion policy : Where the incoming line is placed

Example:

Traditional LRU:
- Victim selection policy → LRU position
- Insertion policy → MRU position

LRU Insertion Policy (LIP):

→ Victim selection policy is LRU
→ Change only in "Insertion Policy".
* Insert everything in LRU position
* Promote to MRU only if it is referenced while in LRU states.
* But no ageing mechanism
  ↳ So Bimodal Insertion Policy (BIP) is proposed.

Bimodal Insertion Policy (BIP):

* Same as LIP but "with low probability" (controlled by Bimodal Throttle parameter) follows a MRU insertion policy.
* Sits like a middle ground between LRU & LIP.
* Has an ageing mechanism while still maintaining thrashing protection properties.
  ↳ But still for an LRU friendly workload, DIP is problematic. So Dynamic Insertion Policy (DIP)

is proposed.

Dynamic Insertion Policy:
* Middle ground between traditional LRU and BIP.
* Tries both LRU and BIP and checks which has the lower miss rate. And uses that.
  ↳ But this causes alot of hardware overload. So "Set Dueling" is proposed.

Zero use lines: Lines not referenced between insertion and eviction

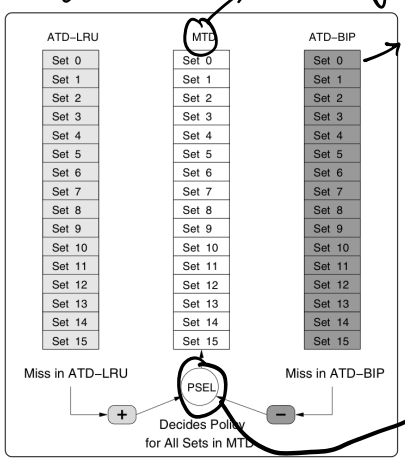
Doubt: "We do not enforce inclusion in our memory model". → Means?
Doubt: "Demand references" → Send exact
Doubt: "Prefetcher pollution".

We only need tag to calculate the statistics

DIP - Global

Main Tag Directory

Only tag → Separate "Auxiliary Tag Directory"!



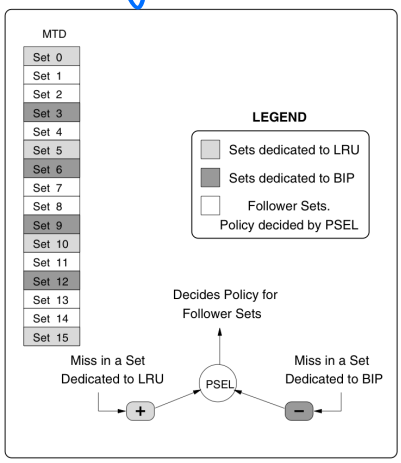
→ Policy Selector

→ Have 2 extra directories
  → one uses BIP → Decreases PSEL by 1
  → another uses LRU → Increases PSEL by 1

+ve → Uses LRU
−ve → Uses BIP

→ Should maintain 2 additional directories ⇒ Overhead!
→ To slightly reduce overhead Dynamic Set Sampling ⇒ cache behaviour can be approx. by a subset.
  ↳ Still high.
  Proposes Set Dueling

Set dueling:



→ Dedicated sets
* Few sets dedicated to LRU
* Few for BIP
→ Follower sets

Instead have ATD we are using the MTD directly

→ But now the question of which sets should be chosen for BIP and LRU.

⇓

Dedicated Set selection Policy:
- Statistically at design time
- Dynamically at runtime

N sets
↓
Divide by (K) → sets dedicated to each policy
↓
Each of the N/K region ⇒ called as constituency.
↓
One set per constituency becomes a dedicated set for each of the policy.

→ Removes the need to separately remember which sets for which policy

Complement-Select Policy

The section introduces a **complement-select policy**, which simplifies the selection process:

- **Set index**: Every set in the cache is indexed using $\log_2(N)$ bits.
- Out of these bits, the most significant $\log_2(K)$ bits identify the constituency, while the remaining bits (the rest of the $\log_2(N/K)$ bits) identify the **offset** within the constituency.

The **complement-select policy** determines which sets are dedicated to which policies:

- **LRU (Least Recently Used)** policy gets sets where the **constituency identifying bits** are equal to the **offset bits**.
- **BIP (Bimodal Insertion Policy)** gets sets where the **offset** is the **complement** of the **constituency identifying bits**.

LRU ⇒ constituency identify bits = offset bits
BIP ⇒ constituency identify bit complement offset bits

Example

For example, if you have a cache with **1024 sets** and want to dedicate **32 sets** to both **LRU** and **BIP**:

- The policy dedicates **set 0 and every 33rd set** to LRU.
- **Set 31 and every 31st set** would go to BIP.

Thus, LRU can be identified by comparing bits in the set index, and similarly, BIP can be identified by using the complement of those bits.

Summary

In this section, a **complement-select policy** is used to allocate sets between two competing policies, **LRU** and **BIP**, without requiring per-set leader markings. The dedicated sets are determined by comparing the constituency identifying bits with the offset within the constituency. This policy is explained in the context of how cache sets are divided and assigned under these competing policies.

2. Constituency and Offset:

Let's say we are dividing the sets into **32 constituencies**. The number of bits needed to identify the constituency is $\log_2(32) = 5$ bits. This means the first 5 bits of the 10-bit set index represent the **constituency**.

The remaining 5 bits will represent the **offset** within that constituency. So, each constituency contains $N/K = 1024/32 = 32$ sets.

3. Equal and Complement Rules:

The **equal** and **complement** rules determine how sets are dedicated to different policies:

- **Equal**: The **LRU (Least Recently Used)** policy will get all sets where the **offset** bits are equal to the **constituency** bits.
- **Complement**: The **BIP (Bimodal Insertion Policy)** will get all sets where the **offset** is the **complement** of the **constituency** bits.

Example of the Equal Rule:

Let's take **constituency 3**. The binary representation of **3** is **00011**.

- The **equal rule** would dedicate any set where the **offset** is also **00011** (binary representation of 3) to the **LRU policy**.
- Therefore, set **0001100011** (which is set 99 in decimal) would be dedicated to LRU because its **constituency bits** (00011) are equal to its **offset bits** (00011).

Example of the Complement Rule:

Now let's look at the **complement rule** for **BIP**. The **complement** of a binary number is simply flipping all the bits (0s become 1s and vice versa).

- For **constituency 3** (which is **00011**), the complement of **00011** is **11100**.
- So, any set where the **offset** is **11100** would be dedicated to **BIP**.
- Therefore, set **0001111100** (which is set 124 in decimal) would be dedicated to BIP because its **constituency bits** (00011) are the complement of its **offset bits** (11100).

Applying It in a Real Example:

- Let's say we have a cache with **1024 sets** and we're dividing them into **32 constituencies**.
- For **constituency 3** (binary 00011):
  - The **equal rule** dedicates set **99** (0001100011) to **LRU**.
  - The **complement rule** dedicates set **124** (0001111100) to **BIP**.

Thus, the **equal rule** means the **offset bits** match the **constituency bits**, and the **complement rule** means the **offset bits** are the bitwise complement of the **constituency bits**.

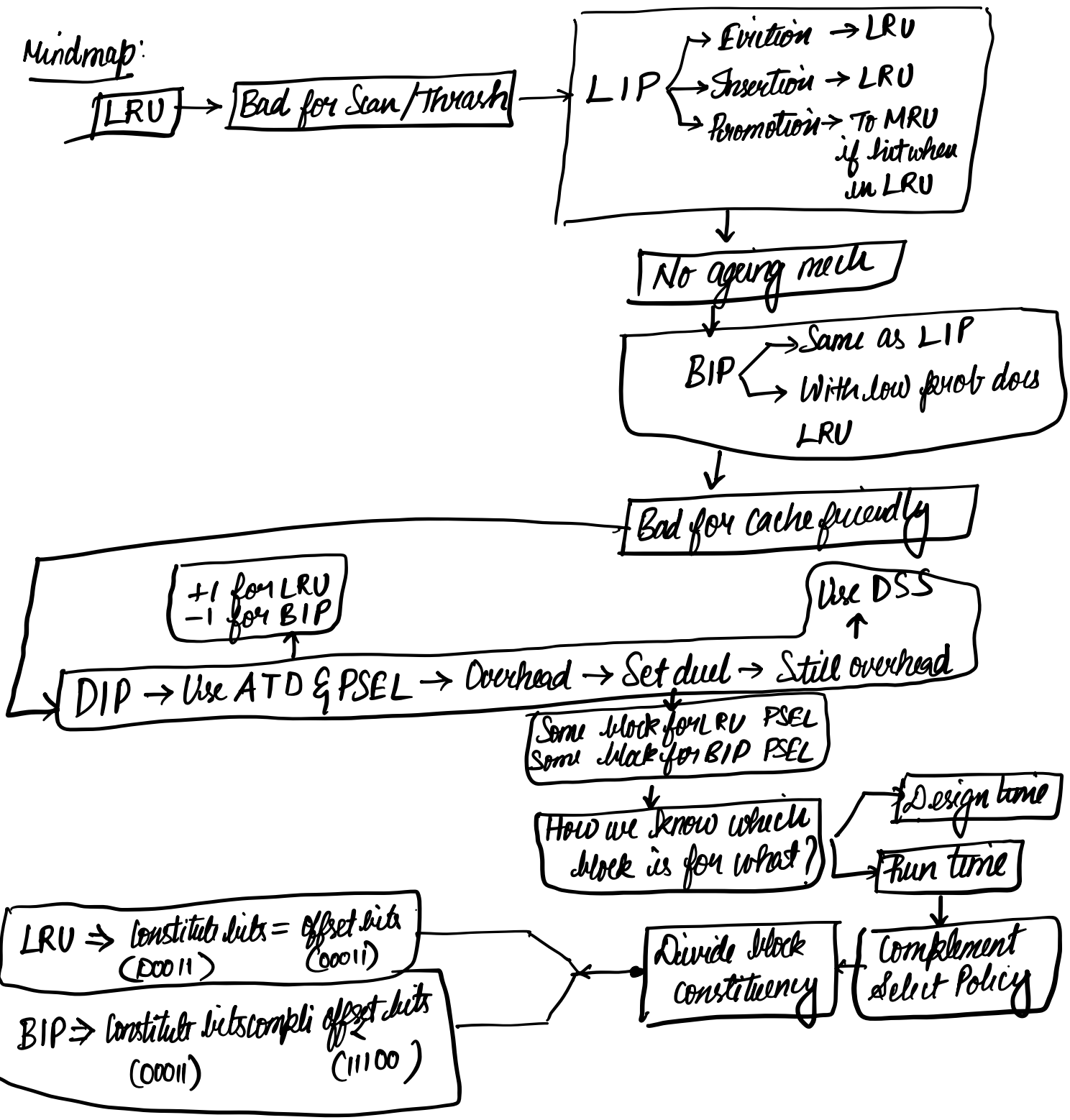
Summary:

- **Equal Rule**: Assigns a set to a policy if the **offset bits** are the same as the **constituency bits**.
  - Example: Constituency 3 (00011) → Set 99 (0001100011) is assigned to LRU.
- **Complement Rule**: Assigns a set to a policy if the **offset bits** are the **complement** of the **constituency bits**.
  - Example: Constituency 3 (00011) → Set 124 (0001111100) is assigned to BIP.

Example

If there are $N/K = 32$ constituency then $\log_2 32 = 5$ bits are constituency bits & remaining are offset.

constituency	offset	
00011	00011	Equal → LRU
00011	11100	→ BIP (Complement)


Mindmap:
LRU
Bad for Scan/Thrash
LIP
Eviction → LRU
Insertion → LRU
Promotion → To MRU if hit when in LRU
No ageing meth
BIP
Same as LIP
With low prob does LRU
Bad for cache friendly
DIP → Use ATD & PSEL → Overhead → Set duel → Still overhead
Use DSS
+1 for LRU
-1 for BIP
Some block for LRU PSEL
Some block for BIP PSEL
How we know which block is for what?
Design time
Run time
Complement Select Policy
Divide block constituency
LRU ⇒ constituency bits = offset bits
(00011) (00011)
BIP ⇒ constituency bits compli offset bits
(00011) (11100)