SNIA DEVELOPER CONFERENCE

SDC 24

BY Developers FOR Developers

September 16-18, 2024
Santa Clara, CA

Efficient Media Utilization Across Dissimilar Cloud Storage Systems

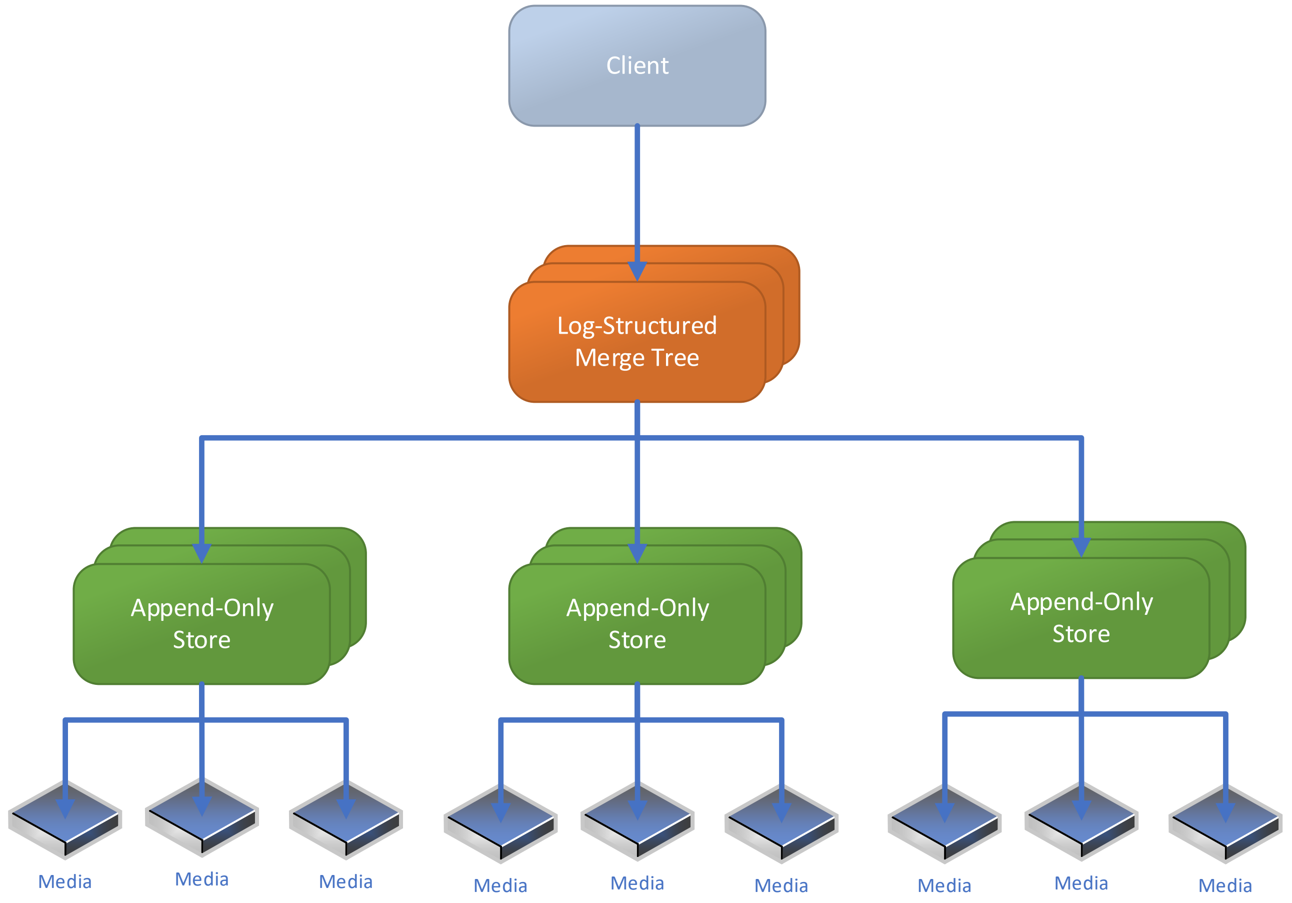
Garret Buban and Madhav Pandya
Microsoft Partner Architects - Azure Storage

Agenda

- Background Information
- Effects of Layering
- New Media Types
- New Object Store
- New Development Platform
- Conclusion

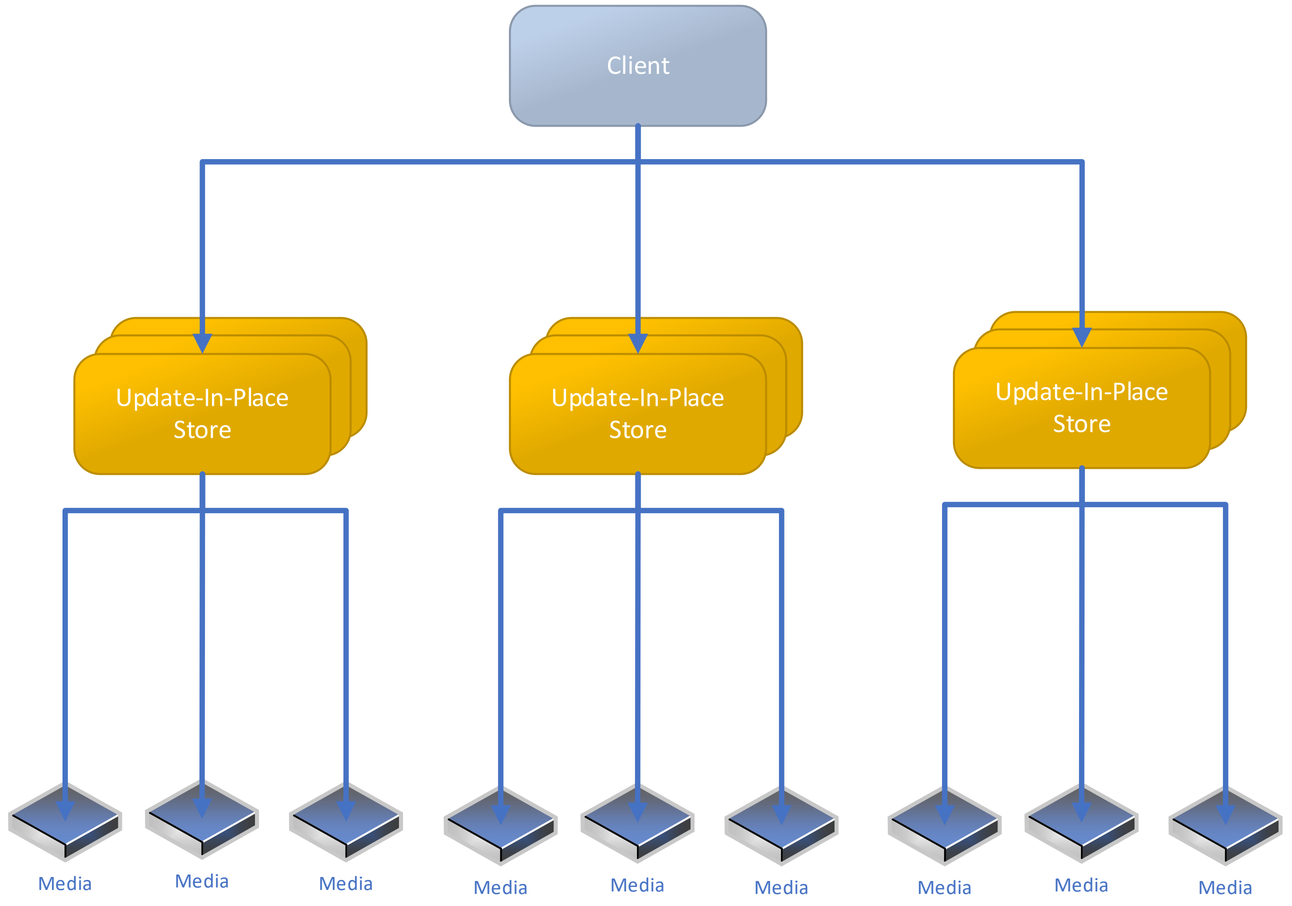
Background Information

Append-Only Store (Key/Value, Files, Etc.)



- Azure Storage ACM SOSP Paper (link), Presentation (link)

Update-in-Place Store (Virtual Disks)

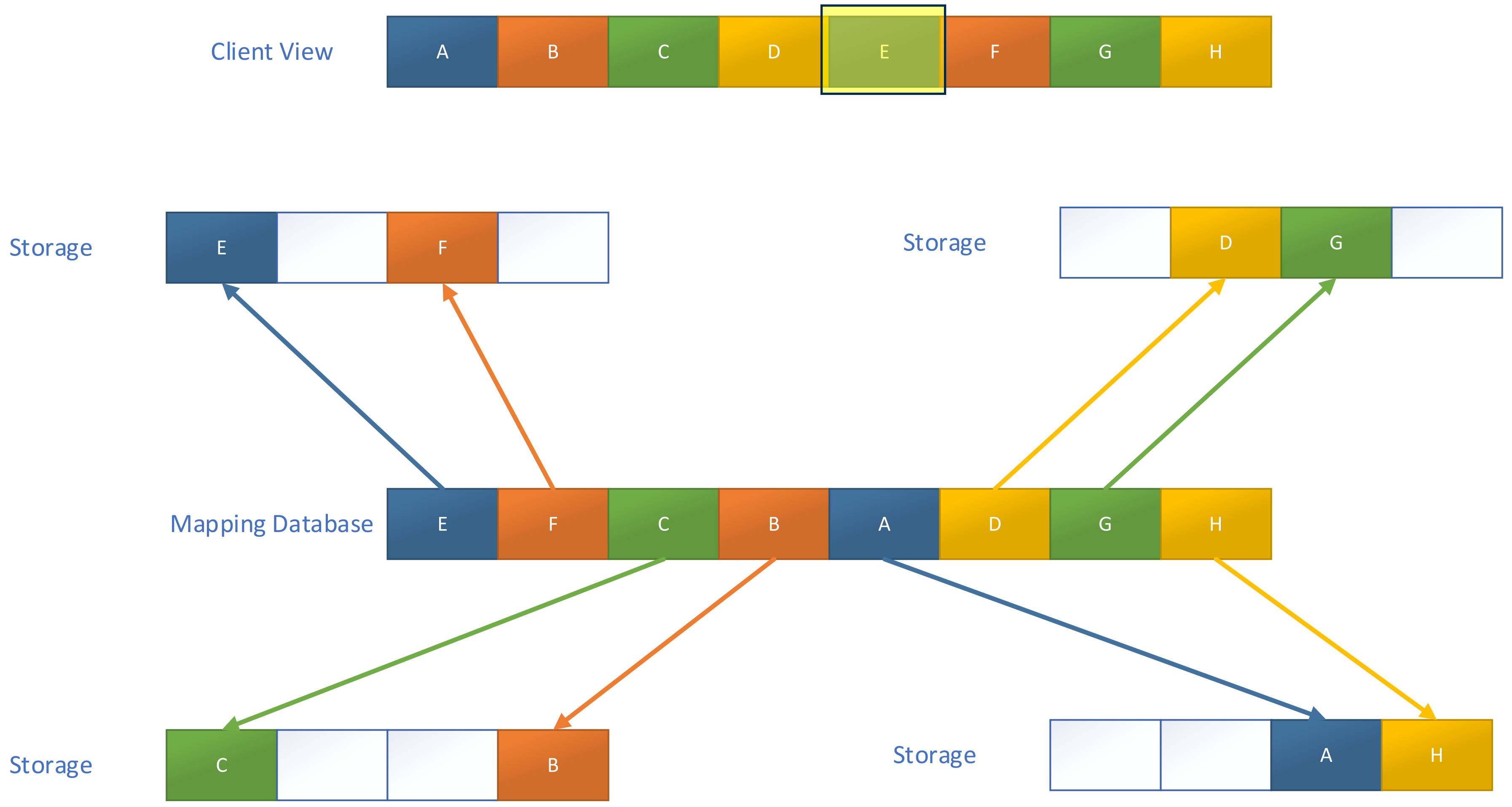


- Direct Drive - Azure's Next-generation Block Storage Architecture (link)

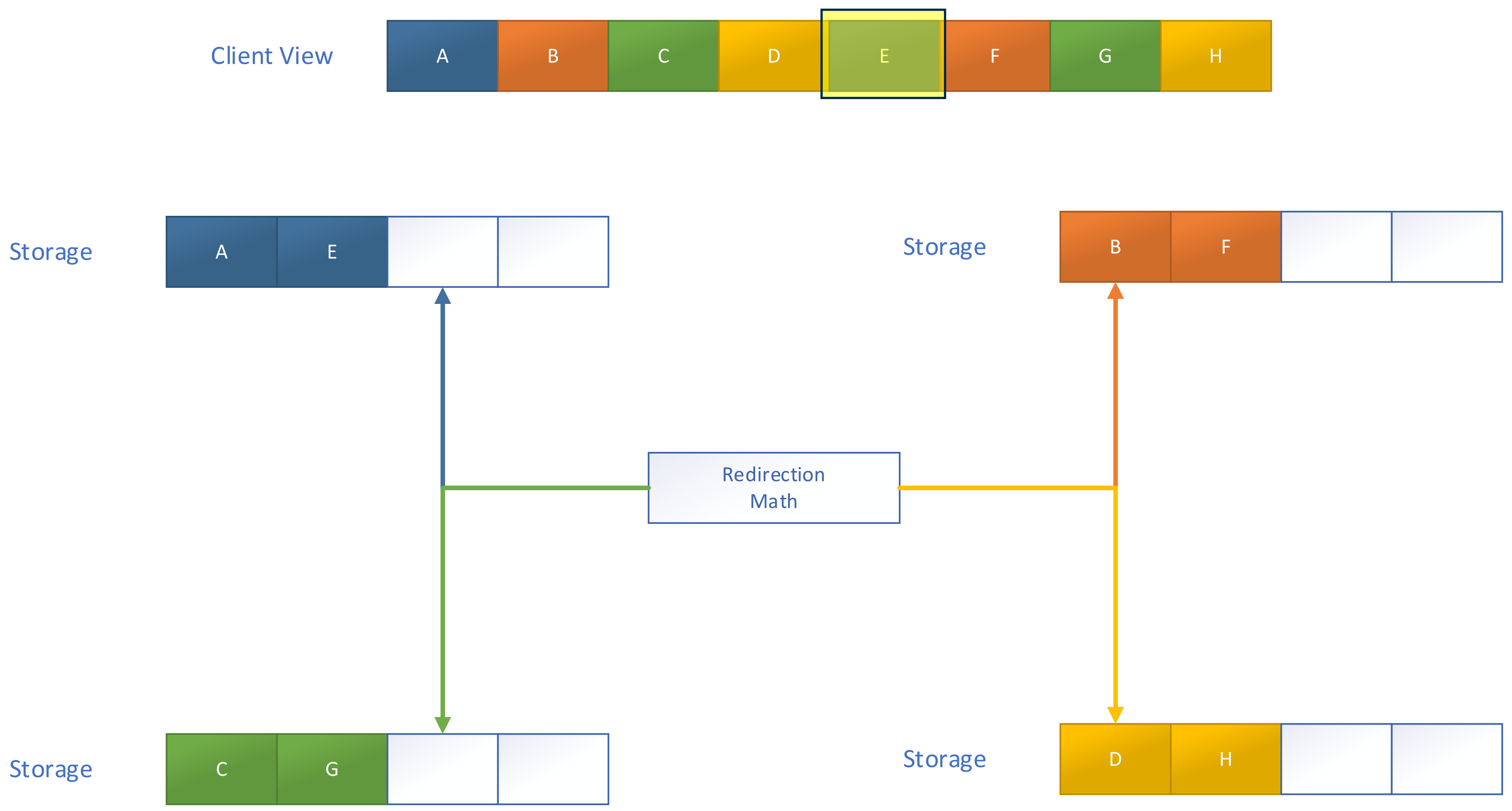
Indirection versus Redirection

- Indirection
  - Dynamic
  - Requires additional storage and updates
  - Requires a centralized “brain”
  - Freedom to place data “anywhere”
- Redirection
  - Static
  - Just math
  - Allows for shared understanding of data placement
  - Forces data location

Indirection

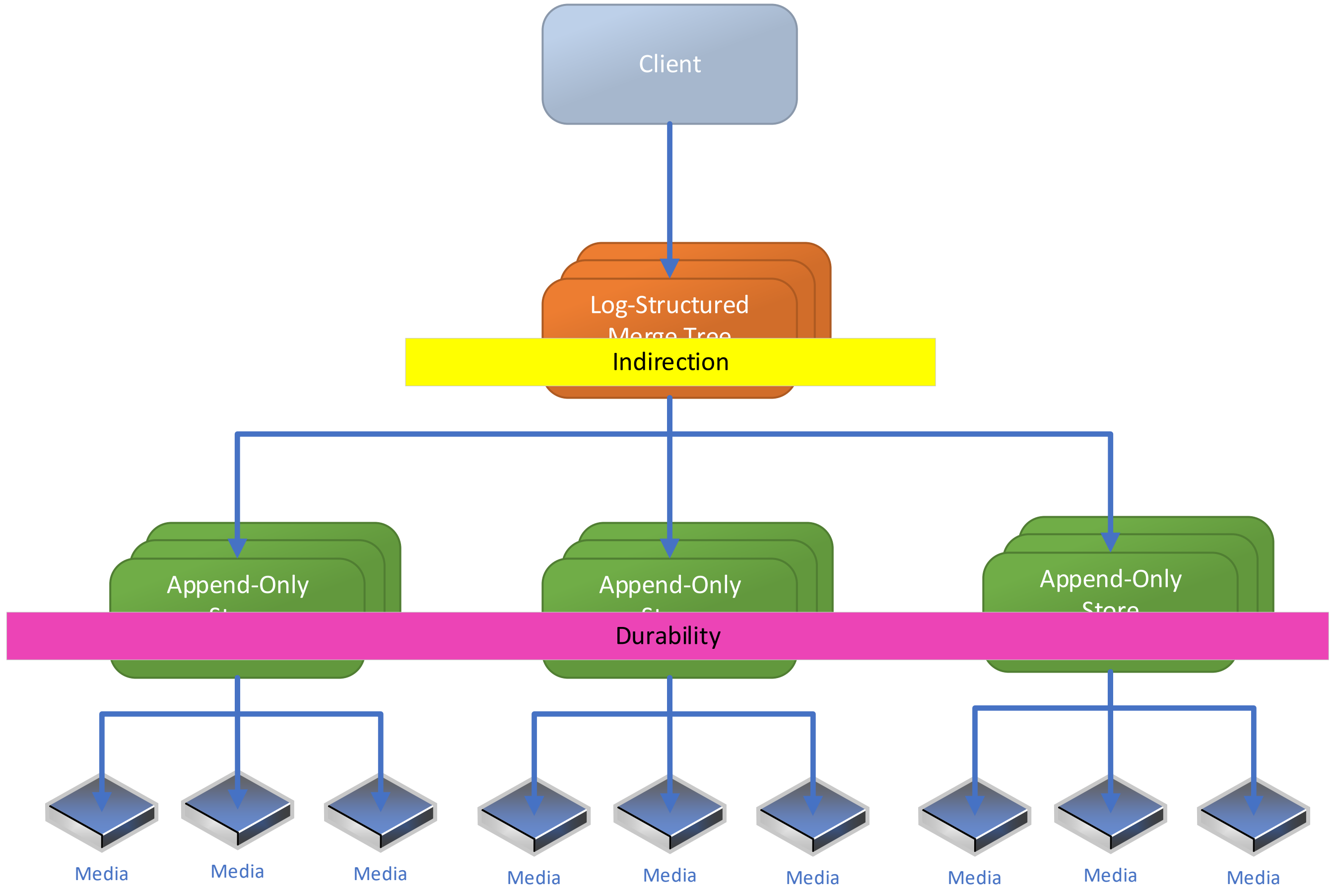


Redirection

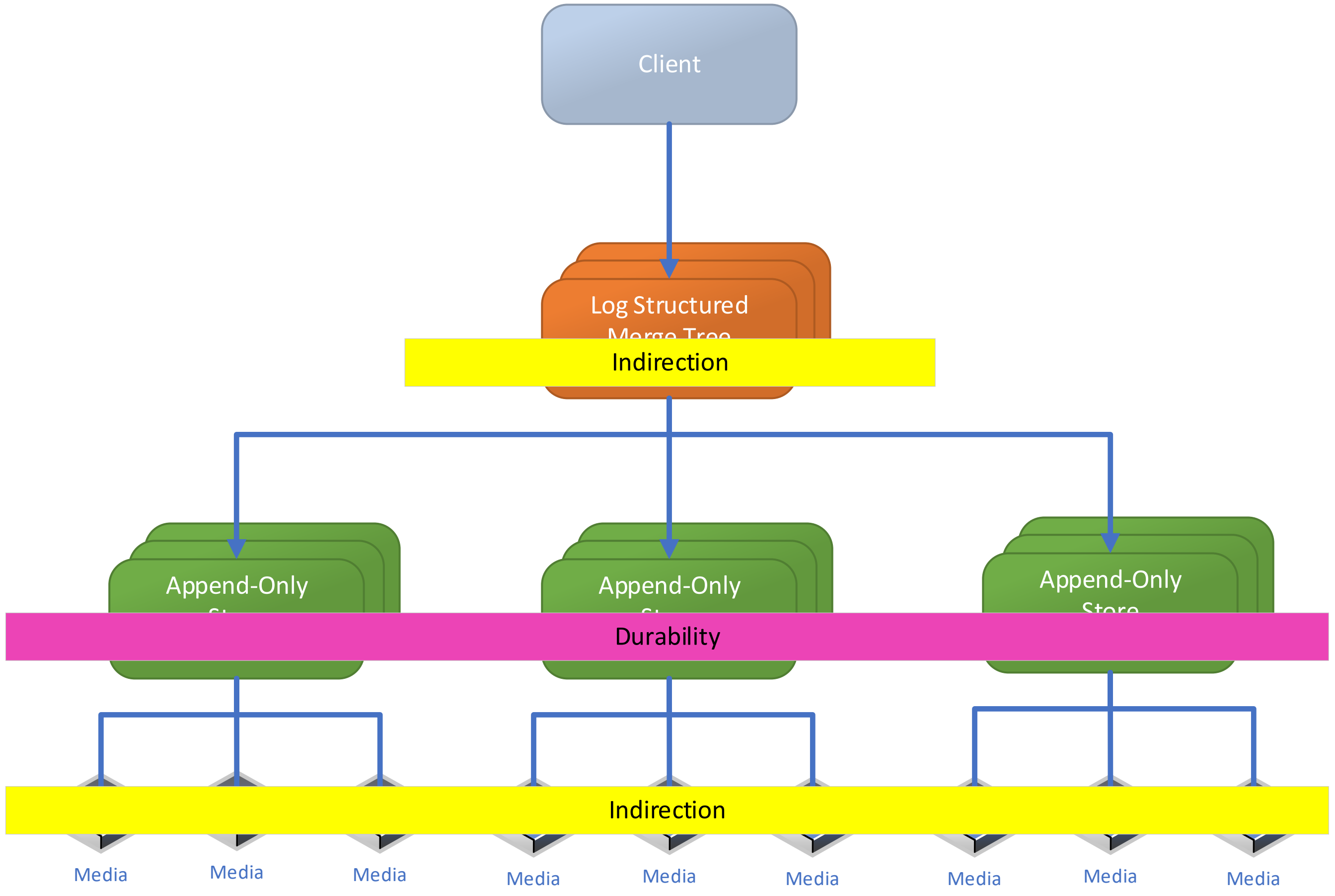


Effects of Layering

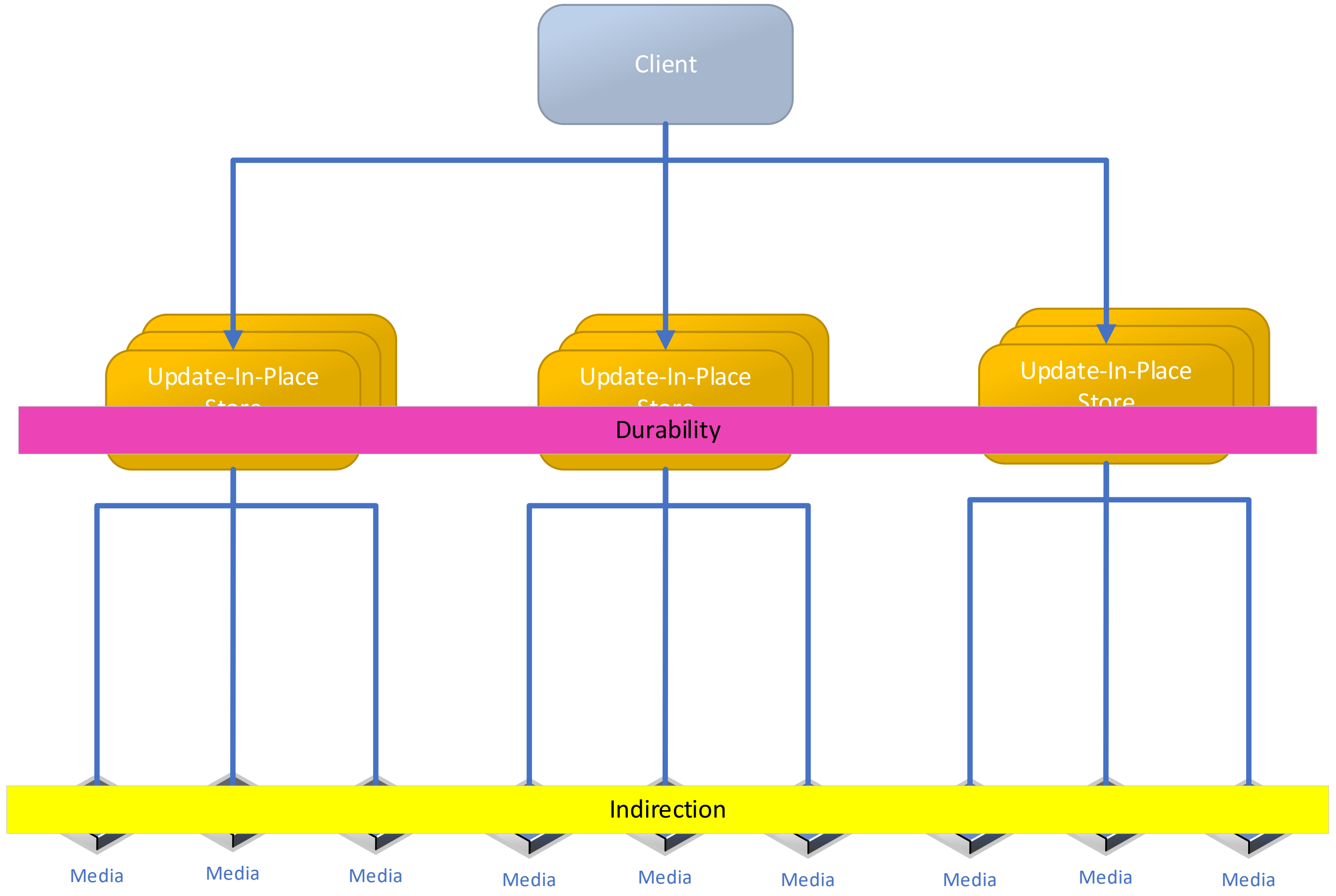
Indirection Above Durability



Rats…


Client
Log Structured Merge Tree
Indirection
Append-Only Store
Append-Only Store
Append-Only Store
Durability
Indirection
Media
Media
Media
Media
Media
Media
Media
Media
Media

Durability Above Indirection

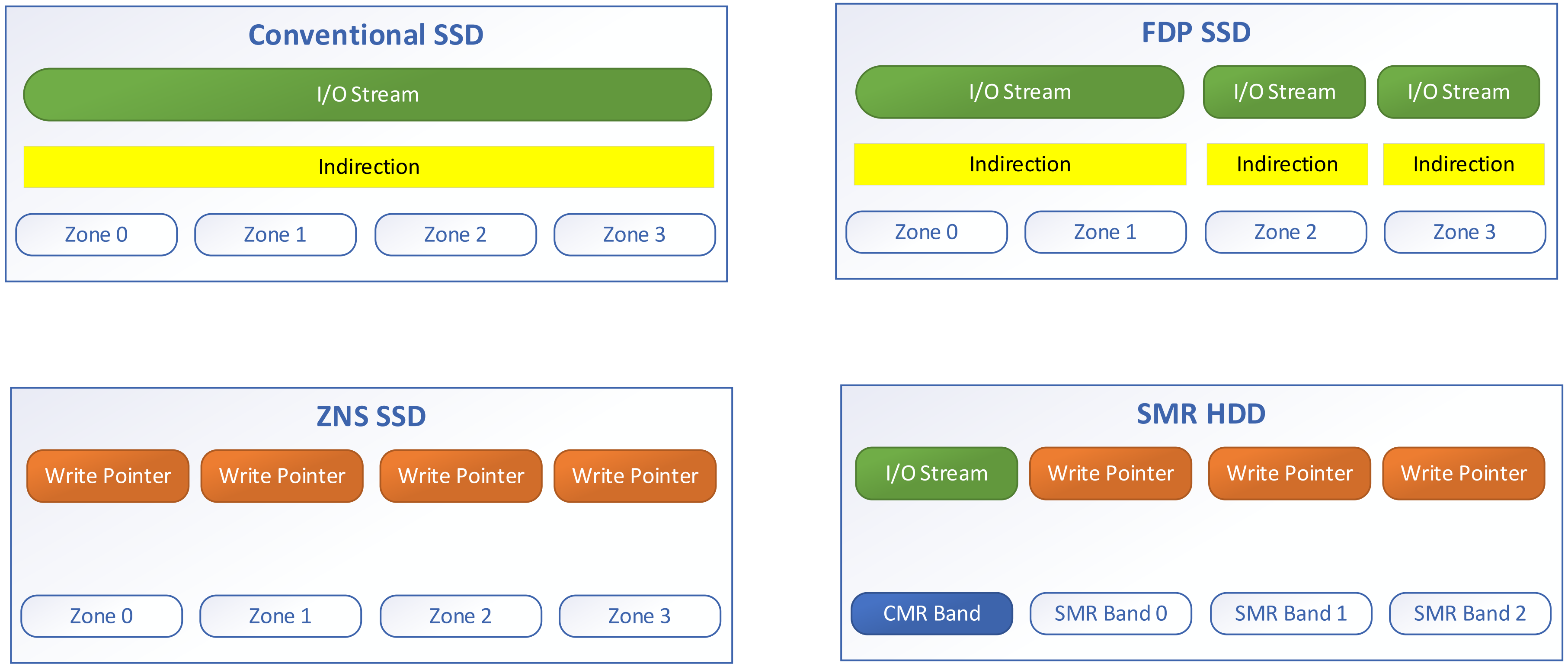


Indirection Location

- Indirection on top of durability
    - Lends itself to append-only
        - Efficient snapshots
    - Single “brain”
        - Reduces performance
        - Creates a single point of failure
    - Easier costs reduction with things like erasure coding
- Durability on top of indirection
    - Lends itself to update-in-place
        - Inefficient snapshots
    - Direct access to data provides highest performance
    - Less flexibility of data placement
- Multiple layers of indirection greatly increases WAF

New Media Types

Overview of Media



Zoned Namespaces SSD

- Less RAM and overprovisioning
  - Saves cost
- Requires an indirection layer above it
  - Adds cost
- Scale is an important factor
- Allows for specialized indirection

Shingled Magnetic Recording (SMR)

- Very similar to ZNS
- Conventional area does not require indirection
    - No need for RAM
    - No need for overprovisioning
    - Allows for easier implementation of a traditional file system
- Shingled area requires indirection*
    - Major cost savings once you have this layer developed

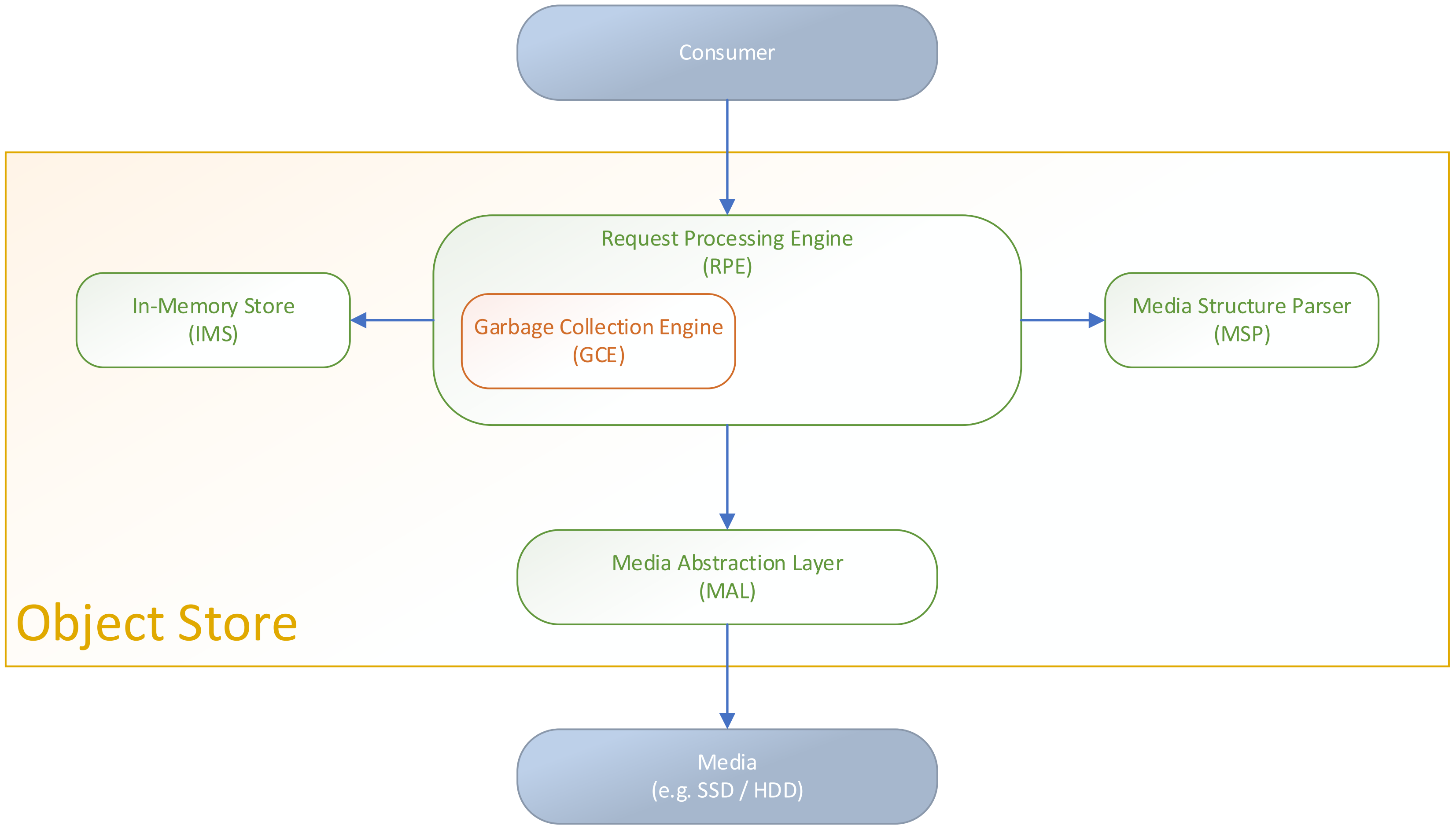
*Host managed assumed due to cost

New Object Store

Specialized Object Store

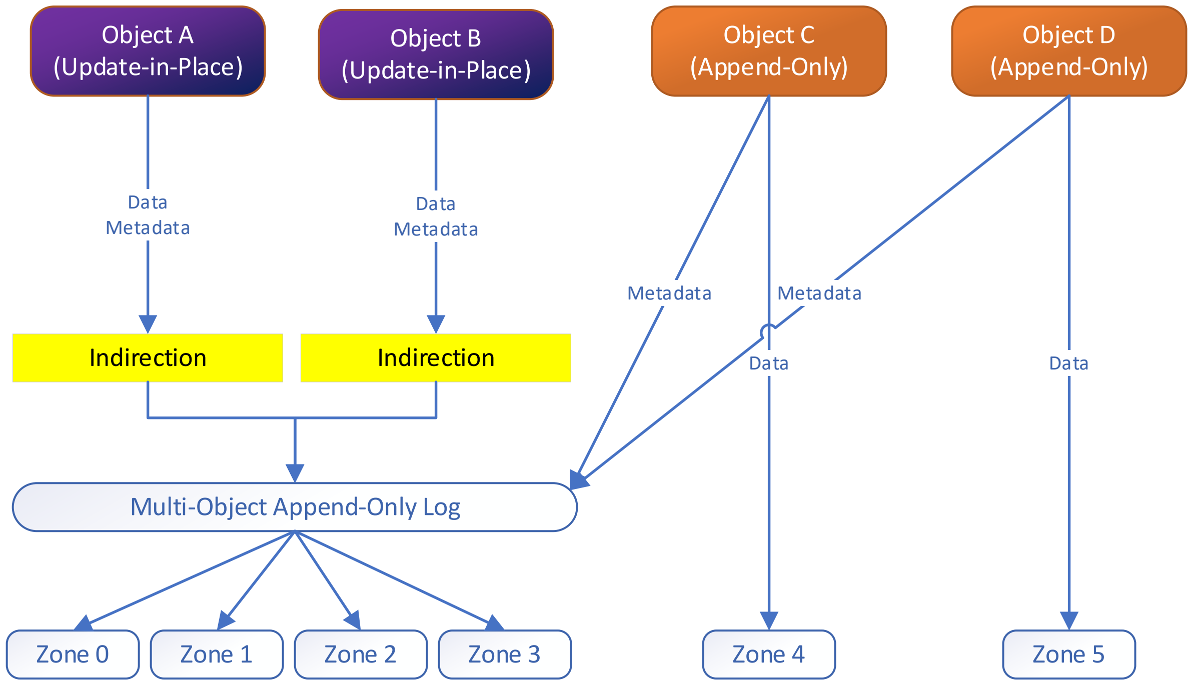
- Provides both Update-in-Place and Append-Only Objects
- Built on ZNS and SMR
  - Also works with conventional devices
  - Abstracts media types
- Provides optional indirection on ZNS/SMR
  - Reducing WAF requires a single layer of indirection
- Supports compression natively
- Minimal feature set
  - No namespace
  - Single consumer

High-level Architecture



Consumer
Request Processing Engine (RPE)
Garbage Collection Engine (GCE)
In-Memory Store (IMS)
Media Structure Parser (MSP)
Media Abstraction Layer (MAL)
Object Store
Media (e.g. SSD / HDD)

Optional Indirection



New Development Platform

Work Unit Based Development

- Inspired by DPU
- Similar to SPDK/DPDK
- Abstracts underlying platform
  - Allows for single code-base
  - Generates optimized code for each platform
    - Mutual exclusion can be implemented in different ways
- Forces developer to break up logic into pipeline stages
  - Tries to strike a balance between coarse vs fine-grain locks
  - Single level of mutual exclusion means no deadlocks
- Designed for massive parallelism

Linear Coding

```
WriteToObject()
{
    Lock(Object);
    CreateWriteRecord(Buffer);
    Lock(Device);
    Offset = Device->GetNextWriteOffset(Length);
    UpdateRecordWithOffset(Buffer, Offset);
    Device->Write(Buffer);
    Unlock(Device);
    UpdateObjectAfterWrite();
    Unlock(Object);
}
```

Linear Coding Improved

```
WriteToObject()
{
    Lock(Object);
    CreateWriteRecord(Buffer);
    Lock(Device);
    Offset = Device->GetNextWriteOffset(Length);
    UpdateRecordWithOffset(Buffer, Offset);
    Unlock(Object);
    Device->Write(Buffer);
    Unlock(Device);
    Lock(Object);
    UpdateObjectAfterWrite();
    Unlock(Object);
}
```

Work Unit Based Coding

```
WriteToObject()
{
    CreateWriteRecord(Buffer);
    ScheduleWork(UpdateRecordWithOffset, ...);
    ScheduleWork(Device->GetWriteOffset, ...);
}
```

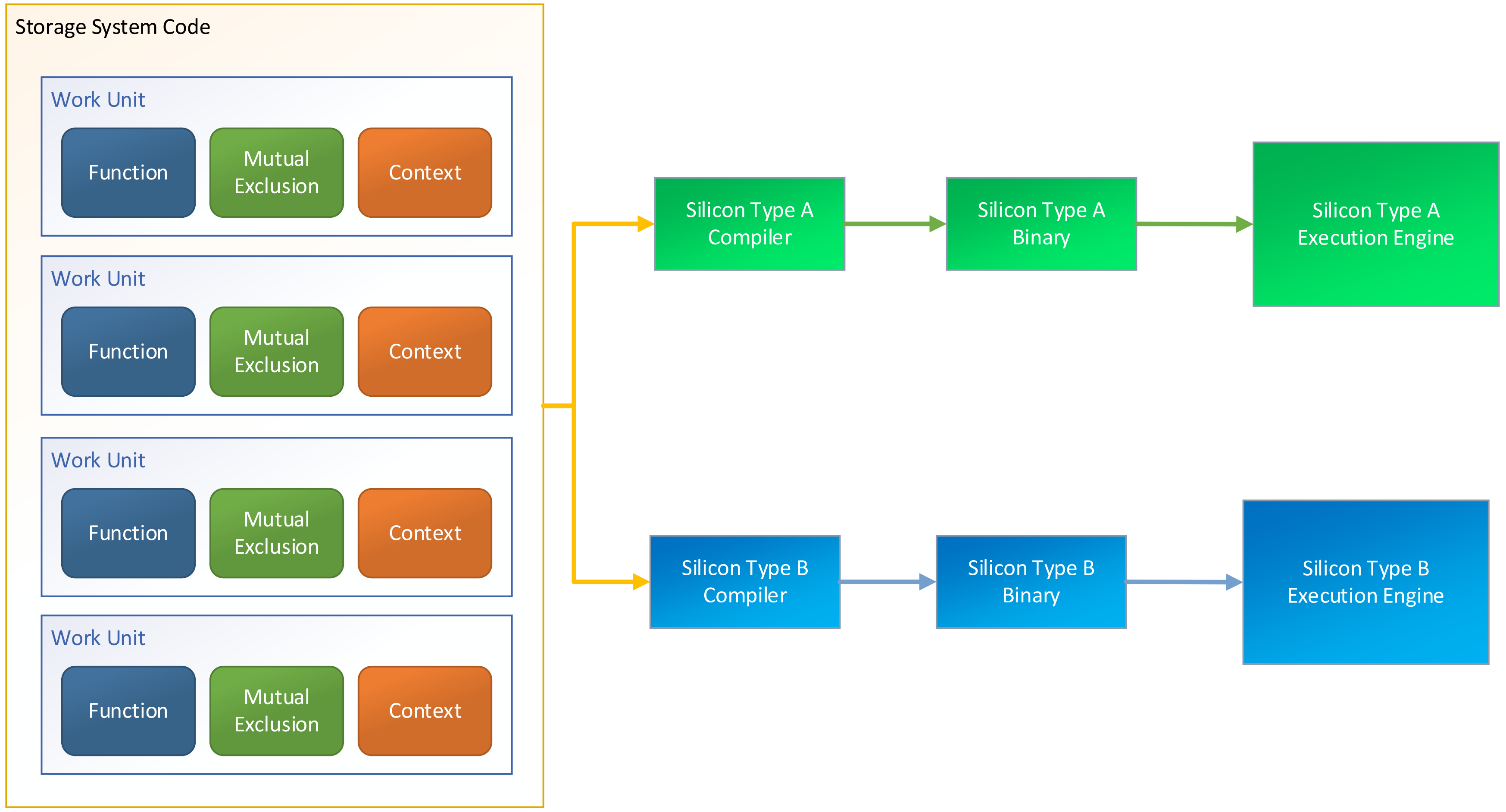
```
UpdateRecordWithOffset(Buffer, Offset)
{
    Buffer->Offset = Offset;
    ScheduleWork(UpdateObjectAfterWrite);
    ScheduleWork(Device->Write, ...);
}
```

```
UpdateObjectAfterWrite()
{
    DoUpdates();
}
```

```
Write(Buffer, Offset)
{
    if(Offset == m_NextOffsetToWrite)
    {
        DoWrites();
    }
    else
    {
        QueueWrite();
    }
}
```

```
GetWriteOffset(Length)
{
    *Offset = m_NextOffsetToUse;
    m_NextOffsetToUse += Length;
}
```

Run-to-Completion Platform

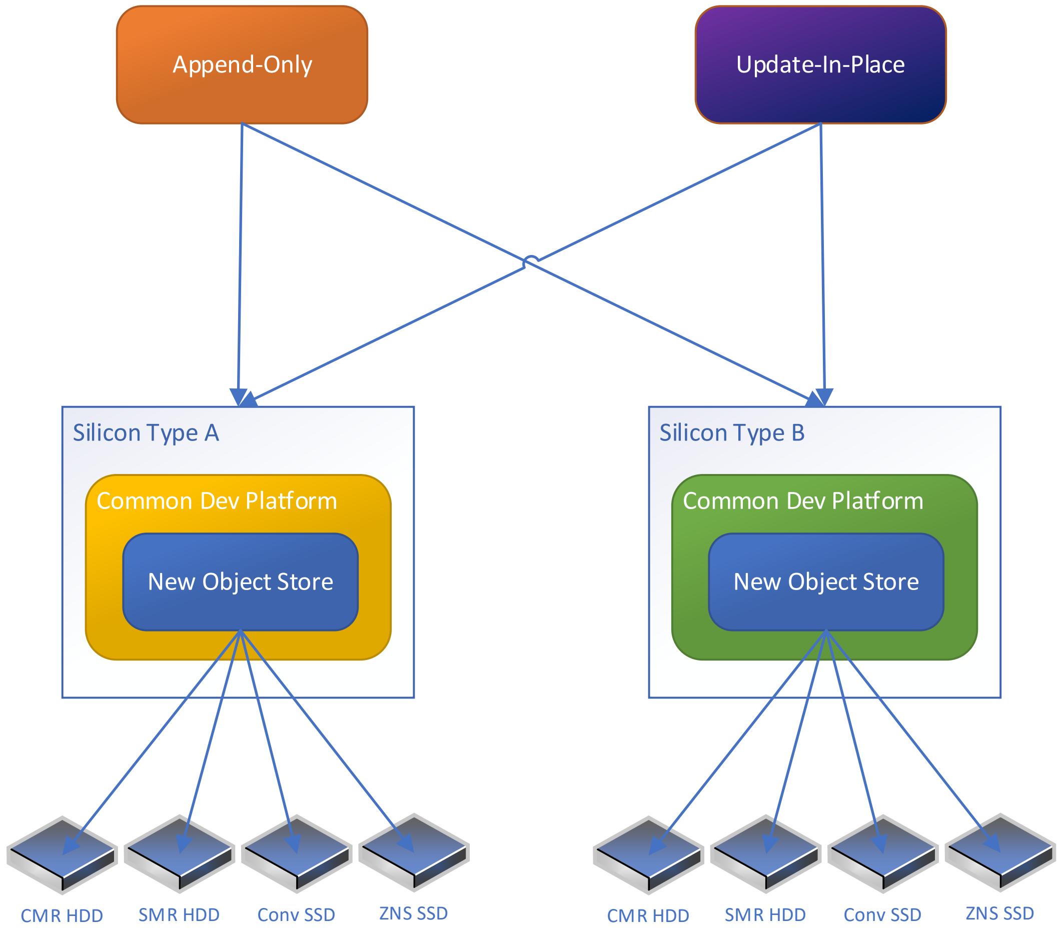


Conclusion

Summary

- Having separate systems provides robust product offerings
- Converging key areas reduces cost
- Abstraction of hardware reduces code duplication

Vision for the Future



Questions?

Please take a moment to rate this session.

Your feedback is important to us.