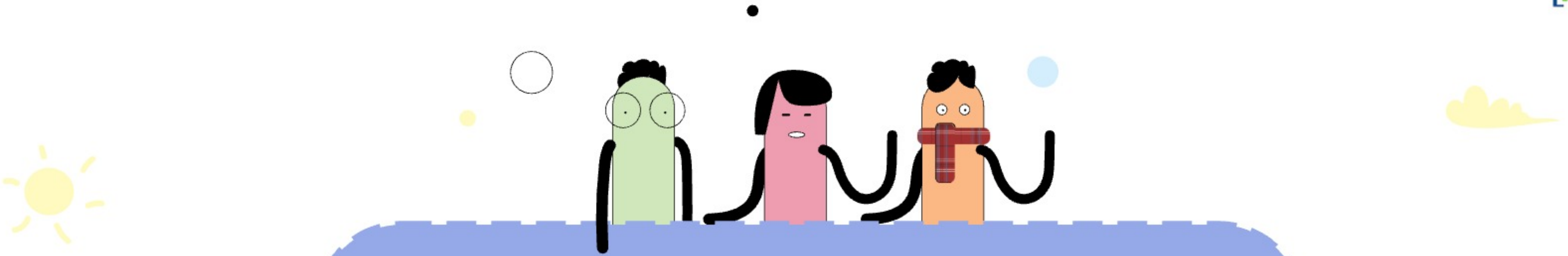
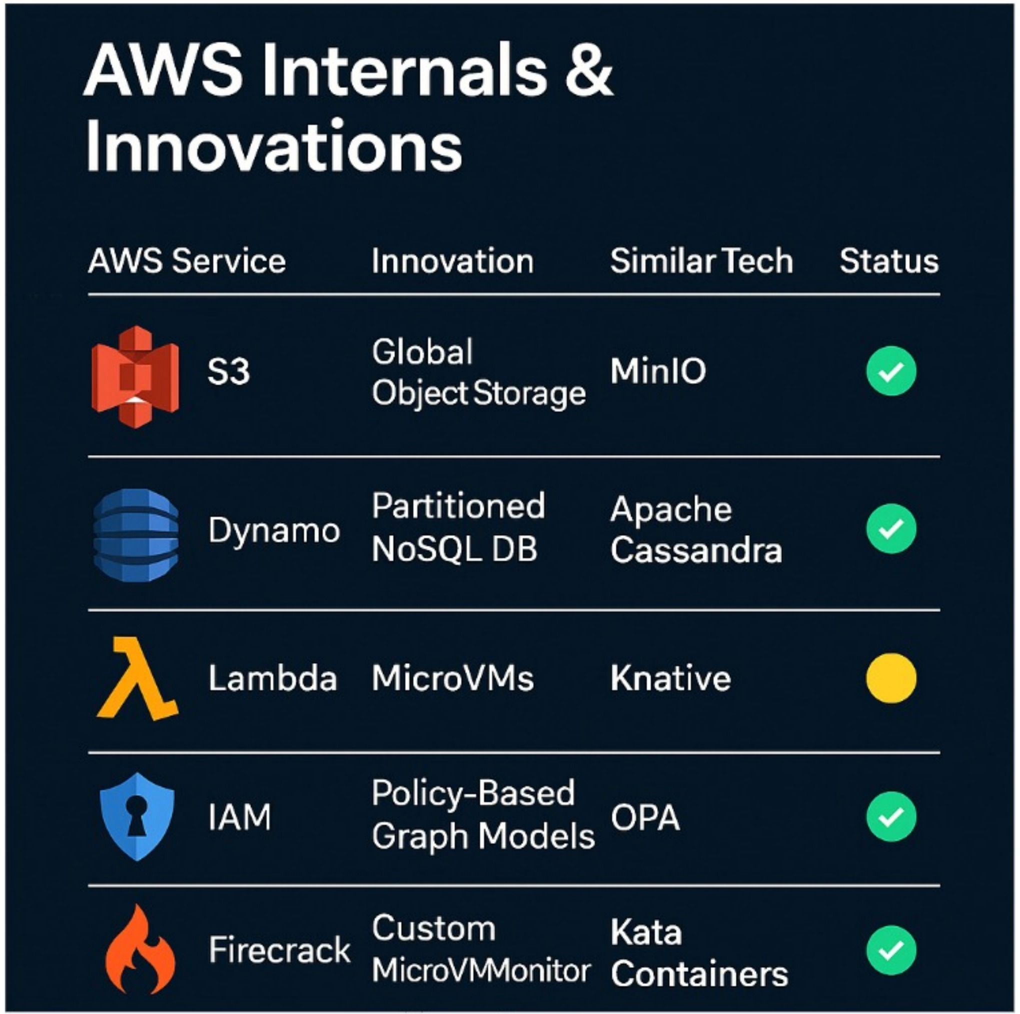


AWS

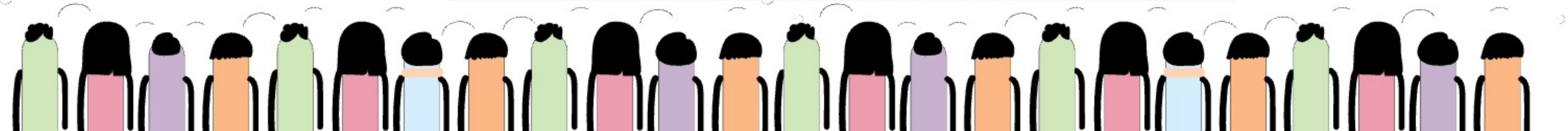
Coderrange

AWS Internal's



AWS Internals & Innovations

AWS Service	Innovation	Similar Tech	Status
S3	Global Object Storage	MinIO	✓
Dynamo	Partitioned NoSQL DB	Apache Cassandra	✓
Lambda	MicroVMs	Knative	●
IAM	Policy-Based Graph Models	OPA	✓
Firecrack	Custom MicroVMMonitor	Kata Containers	✓

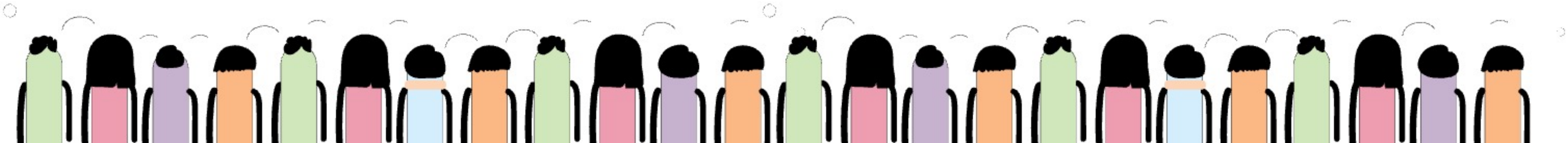


S3 Internals

🧠 Why AWS Built It:

Traditional NAS/SAN could not scale to billions of files or withstand datacenter-wide failures. S3 needed to be:

- **Highly available**
- **Globally scalable**
- **Durable (11 9′s)**



S3 Coding

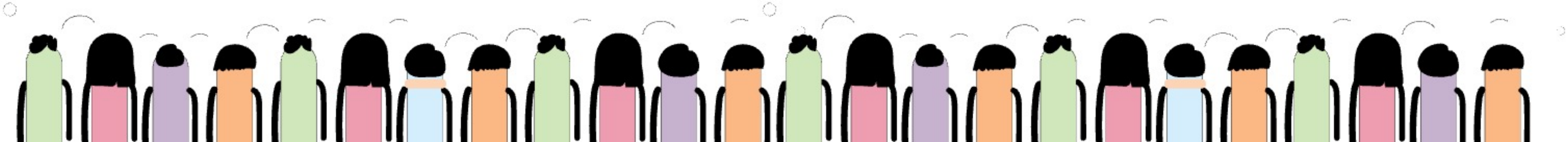
```
for (i = 0; i < 8; i++) {
    unsigned int cp_count;
    unsigned int len = n;
    if (user(group_info[i]){
        return -EFAULT;
    grouplist += NGS_PER_BLOCK;
    count -= cp_count;
```

```
group_info = kmalloc(usr)
if (!group_info)
    return NULL;
group_info->ngroups = gidsetsize;
group_info->nblocks = nblocks;
atomic_set(&group_info->usage, 1)
```

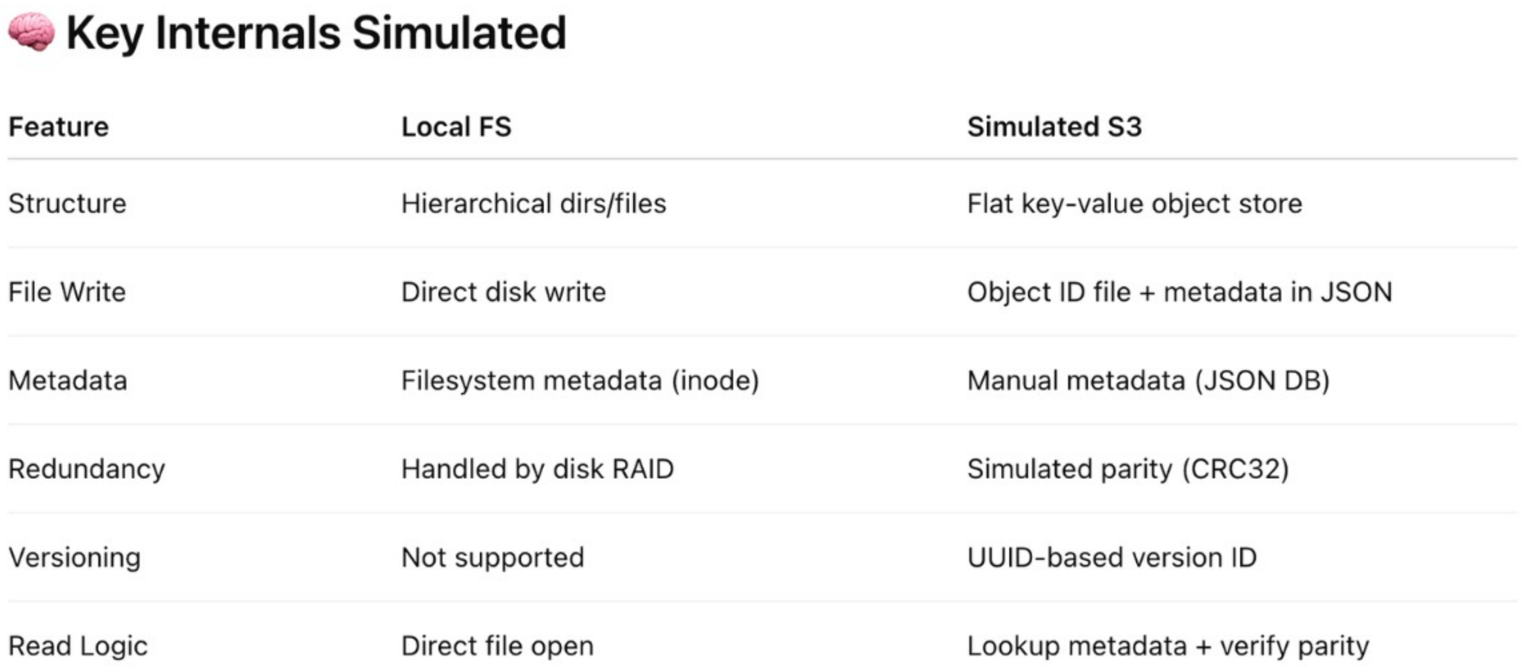
```
if (gidsetsize <= NGROUPS_SMALL)
    group_info->blocks[0] = gr;
else {
    for (i = 0; i < nb; i++) {
        gid_t *b;
        b = (void *)__get(GFP);
        if (!b)
            goto partial_alloc;
        group_info->blo[i] = b;
}
```

```
struct group_inf
int nblocks
```

```
void groups_free(struct*) {
    if (group_info->blocks[0]) {
        int i;
        for (i = 0; i < 1; i++)
            free_page();
    }
    ee(group_info);
```

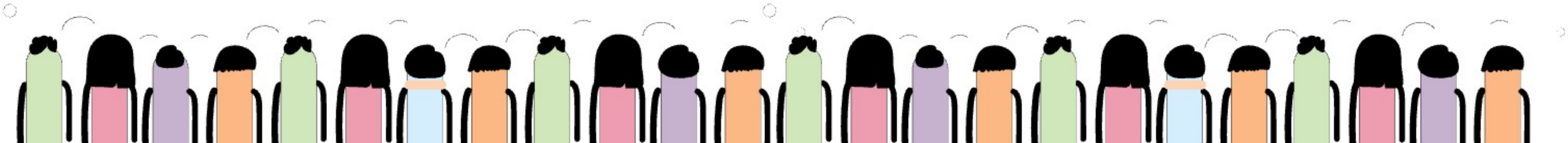


S3 Internals

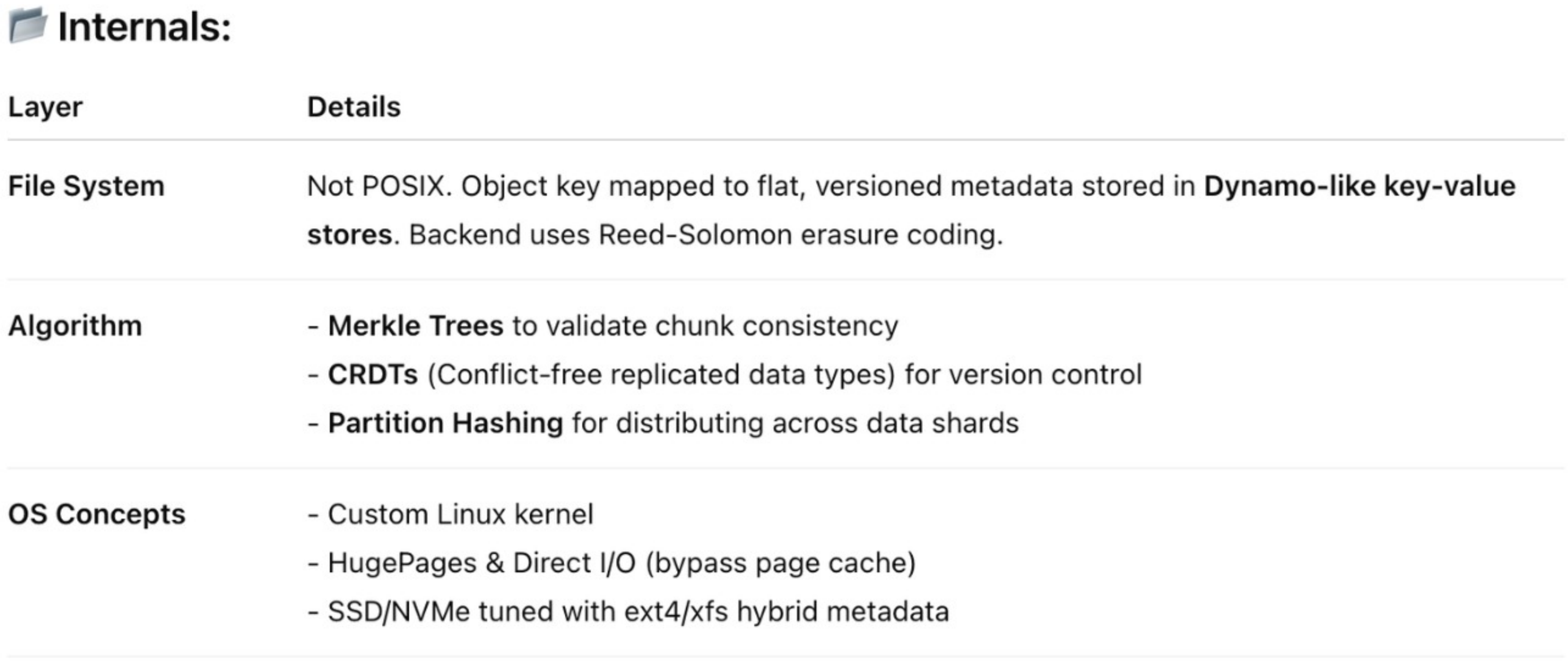


🧠 Key Internals Simulated

Feature	Local FS	Simulated S3
Structure	Hierarchical dirs/files	Flat key-value object store
File Write	Direct disk write	Object ID file + metadata in JSON
Metadata	Filesystem metadata (inode)	Manual metadata (JSON DB)
Redundancy	Handled by disk RAID	Simulated parity (CRC32)
Versioning	Not supported	UUID-based version ID
Read Logic	Direct file open	Lookup metadata + verify parity

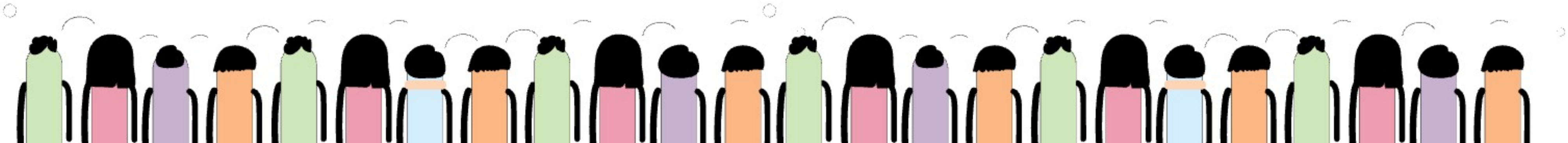


S3 Internals



📂 Internals:

Layer	Details
File System	Not POSIX. Object key mapped to flat, versioned metadata stored in **Dynamo-like key-value stores**. Backend uses Reed-Solomon erasure coding.
Algorithm	- **Merkle Trees** to validate chunk consistency - **CRDTs** (Conflict-free replicated data types) for version control - **Partition Hashing** for distributing across data shards
OS Concepts	- Custom Linux kernel - HugePages & Direct I/O (bypass page cache) - SSD/NVMe tuned with ext4/xfs hybrid metadata

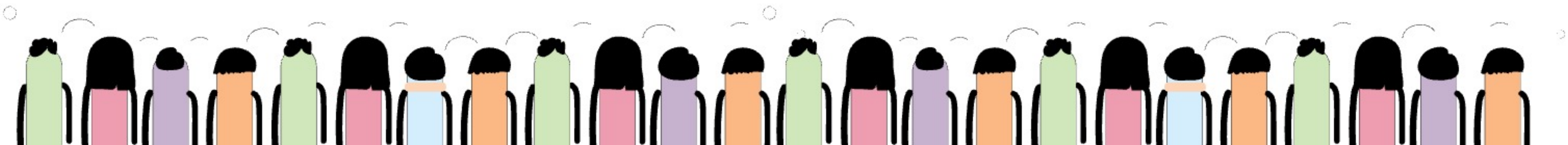


Lambda

⚡ 2. Lambda (Serverless Execution)

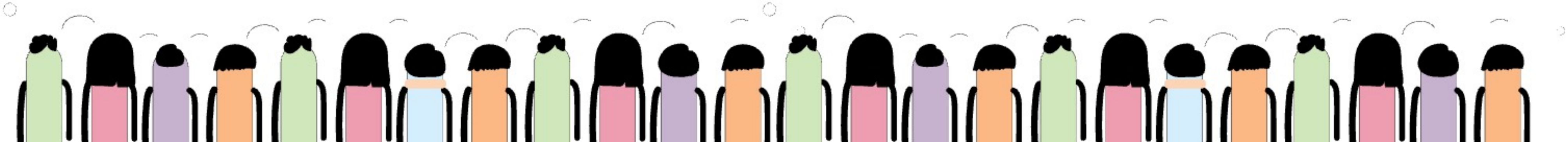
🧠 Why AWS Built It:

Needed **instant compute on demand** without provisioning. Containers were too slow for <50ms starts.

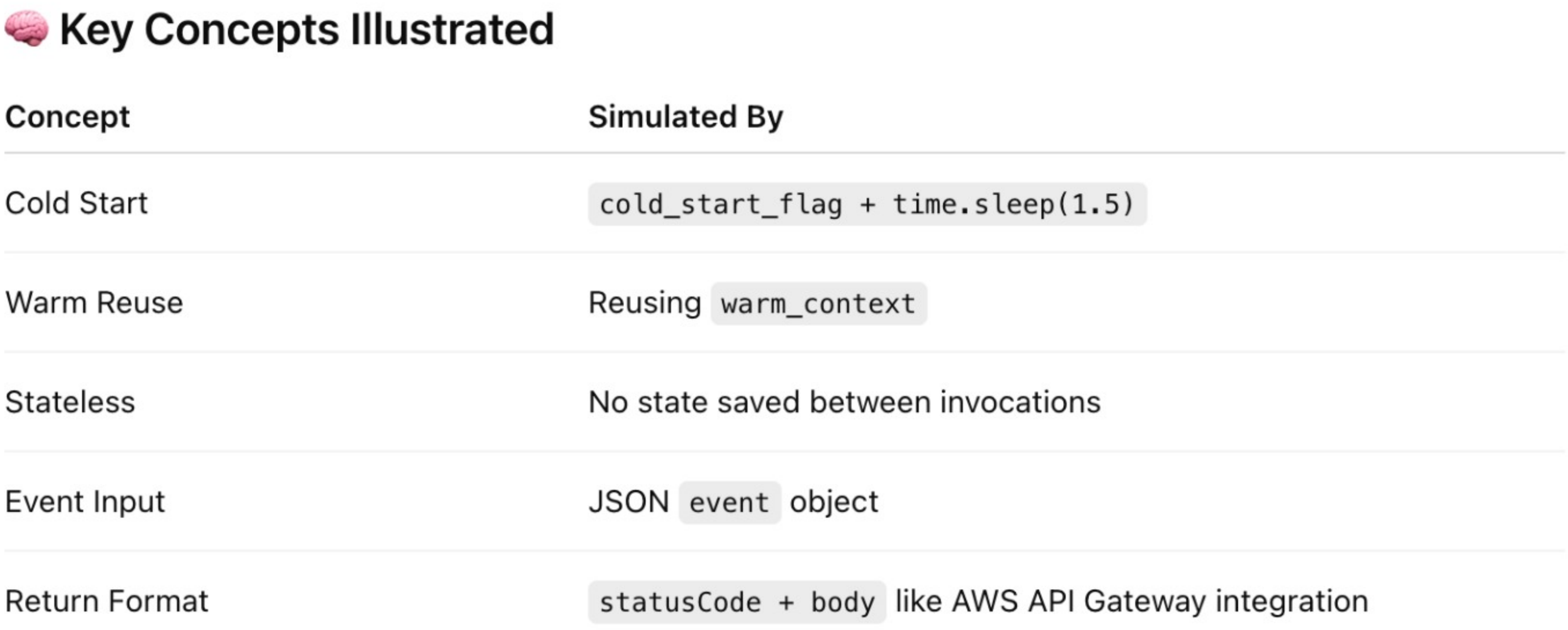


Lambda Coding

for (i = 0; i < 8; i++) {
unsigned int cp_count;
unsigned int len = n;
if (user(group_info[i]){
return -EFAULT;
grouplist += NGS_PER_BLOCK;
count -= cp_count;
group_info = kmalloc(usr)
if (!group_info)
return NULL;
group_info->ngroups = gidsetsize;
group_info->nblocks = nblocks;
atomic_set(&group_info->usage, 1)
if (gidsetsize <= NGROUPS_SMALL)
group_info->blocks[0] = gr;
else {
for (i = 0; i < nb; i++) {
gid_t *b;
b = (void *)__get(GFP);
if (!b)
goto partial_alloc;
group_info->blo[i] = b;
}
struct group_inf
int nblocks
void groups_free(struct*) {
if (group_info->blocks[0]) {
int i;
for (i = 0; i < 1; i++)
free_page();
(group_info);

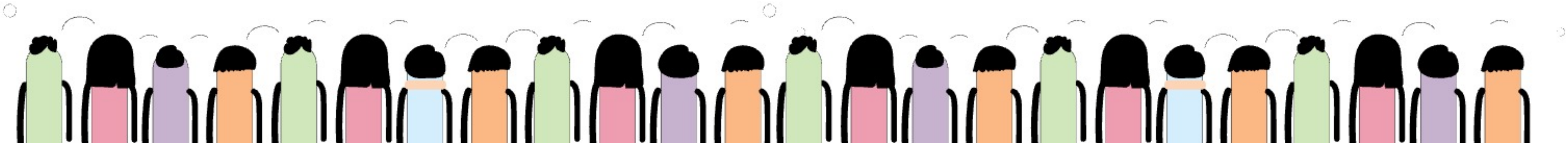


Lambda Coding



🧠 Key Concepts Illustrated

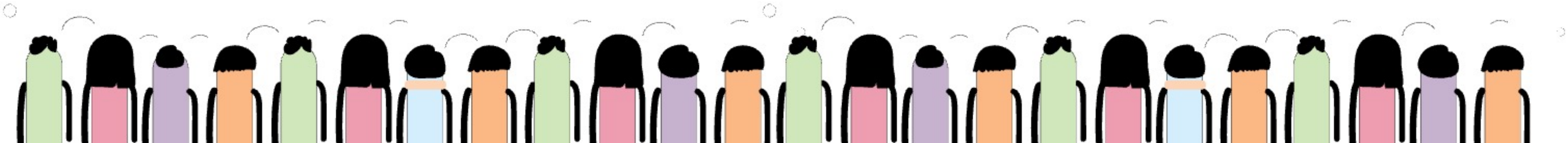
Concept	Simulated By
Cold Start	`cold_start_flag + time.sleep(1.5)`
Warm Reuse	Reusing `warm_context`
Stateless	No state saved between invocations
Event Input	JSON `event` object
Return Format	`statusCode + body` like AWS API Gateway integration



Lambda

📦 Internals:

Layer	Details
File System	Immutable file system based on container image layers, union mount
OS/Runtime	**Firecracker microVM** built with Rust (fast boot + security sandbox) - Uses `KVM` (Kernel-based Virtual Machine) - Boot time < 125ms
Algorithm	Snapshot + Copy-on-Write for fast startup
Cold Start Fix: Provisioned Concurrency pre-warms containers	



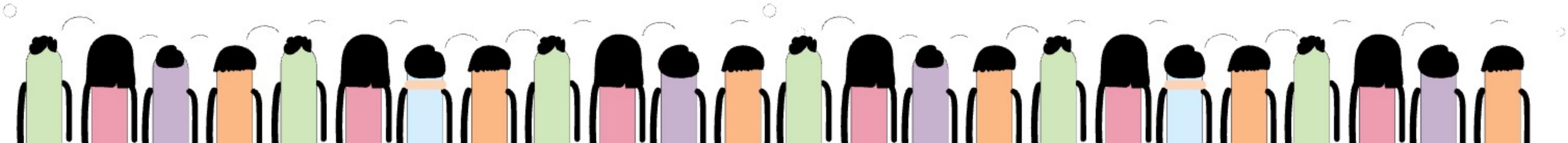
DynamoDB

🧠 Why AWS Built It:

Needed a **highly available, eventually consistent** store. Traditional RDBMS couldn’t scale at Amazon retail level.

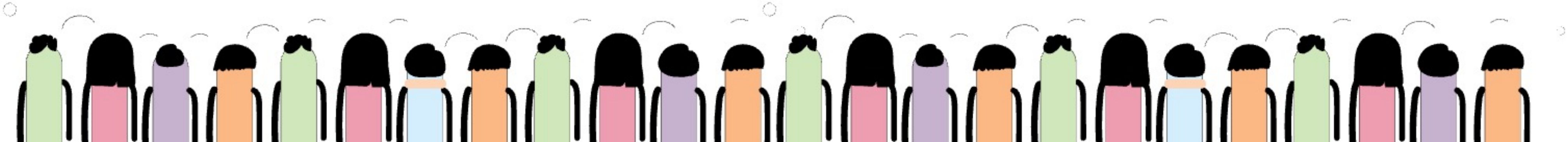
🧱 Internals:

Layer	Details
Data Structure	**LSM Trees** for fast write throughput
Algorithms	- **Quorum-Based Writes** (N, R, W model) - **Gossip Protocols** for node health - **Vector Clocks** to detect write conflicts
File System	Custom SSTable format (like LevelDB) over NVMe
OS-Level	Thread pool tuning, I/O scheduler using deadline/cfq



DynamoDB Coding

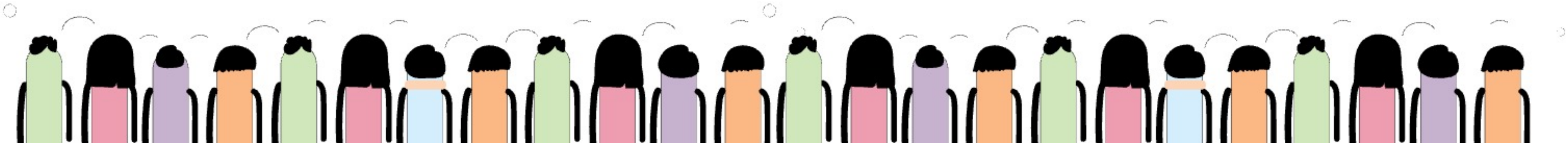
for (i = 0; i < 8; i++) {
unsigned int cp_count;
unsigned int len = n;
if (user(group_info[i]){
return -EFAULT;
grouplist += NGS_PER_BLOCK;
count -= cp_count;
group_info = km alloc(usr)
if (!group_info)
return NULL;
group_info->ngroups = gidsetsize;
group_info->nblocks = nblocks;
atomic_set(&group_info->usage, 1)
if (gidsetsize <= NGROUPS_SMALL)
group_info->blocks[0] = gr;
else {
for (i = 0; i < nb; i++) {
b = (void *)__get(GFP);
if (!b)
goto partial_alloc;
group_info->blo[i] = b;
}
struct group_inf
void groups_free(struct*) {
free_page();



DynamoDB

✅ What is DynamoDB?

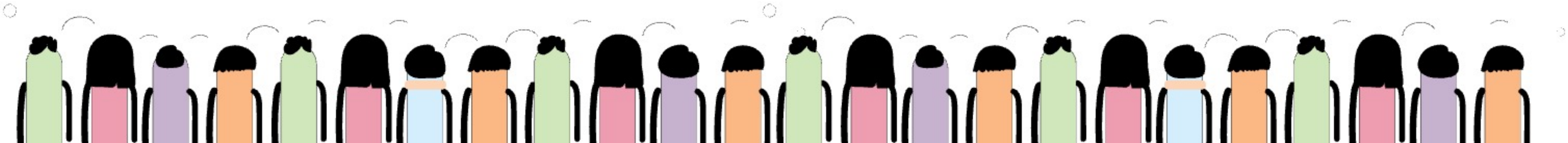
Feature	Description
🗃️ Type	NoSQL (key-value & document)
🧠 Core Idea	Distributed hash table (DHT) + Quorum consistency
🧩 Data Structure	Partition key (required) + Sort key (optional)
📡 Writes	Eventually consistent or strongly consistent
🔁 Replication	Multi-AZ, quorum-based



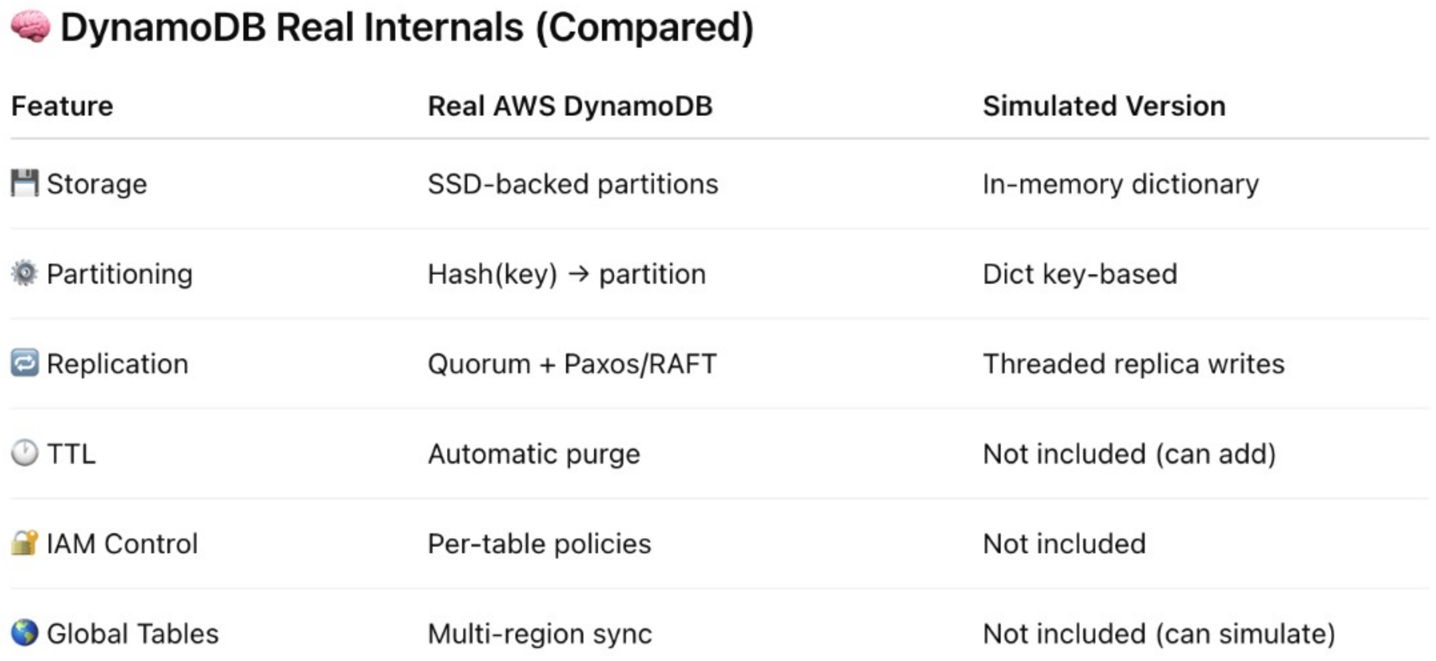
DynamoDB

🧠 What It Simulates

DynamoDB Concept	Python Code Equivalent
Partition Key / Sort Key	2-level dict (`replica[pk][sk]`)
Replication	Writes to 3 in-memory "nodes"
Quorum Read	Majority vote among 3 replicas
Eventual Consistency	No locks across all replicas
Threaded Writes	Simulates async replication latency

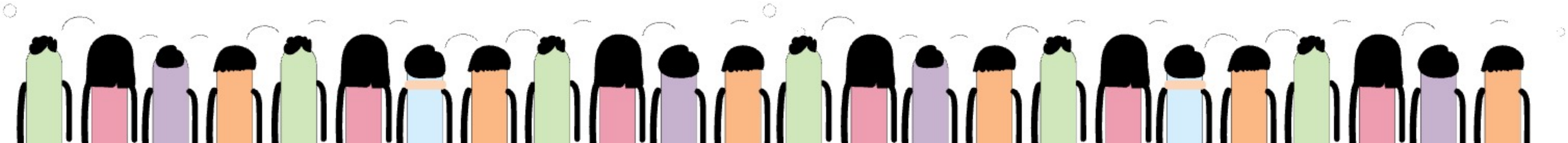


DynamoDB

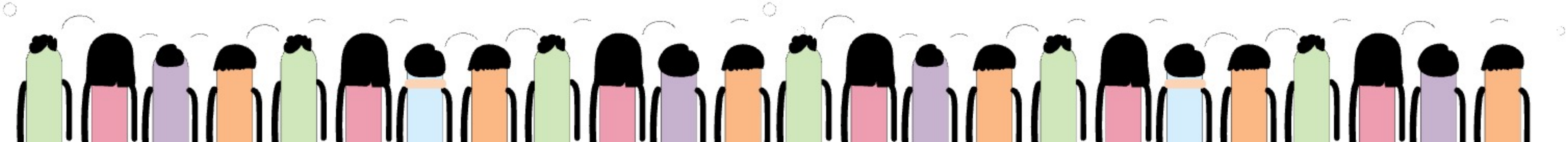


🧠 DynamoDB Real Internals (Compared)

Feature	Real AWS DynamoDB	Simulated Version
💾 Storage	SSD-backed partitions	In-memory dictionary
⚙️ Partitioning	Hash(key) → partition	Dict key-based
🔁 Replication	Quorum + Paxos/RAFT	Threaded replica writes
🕛 TTL	Automatic purge	Not included (can add)
🔐 IAM Control	Per-table policies	Not included
🌎 Global Tables	Multi-region sync	Not included (can simulate)



SQS



SQS Coding

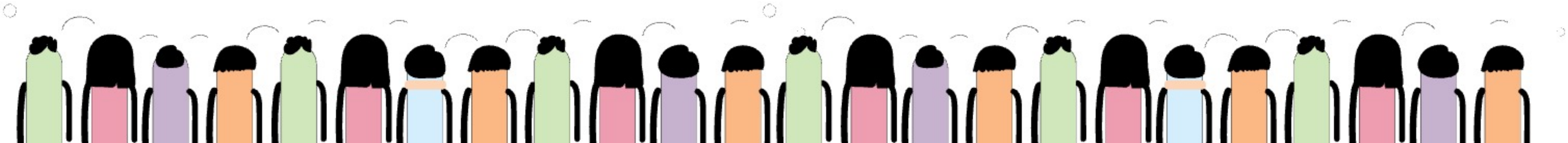
```
for (i = 0; i < 8; i++) {
  unsigned int cp_count;
  unsigned int len = n;
  if (user(group_info[i]){
    return -EFAULT;
  grouplist += NGS_PER_BLOCK;
  count -= cp_count;
```

```
group_info = kmalloc(usr)
if (!group_info)
  return NULL;
group_info->ngroups = gidsetsize;
group_info->nblocks = nblocks;
atomic_set(&group_info->usage, 1)
```

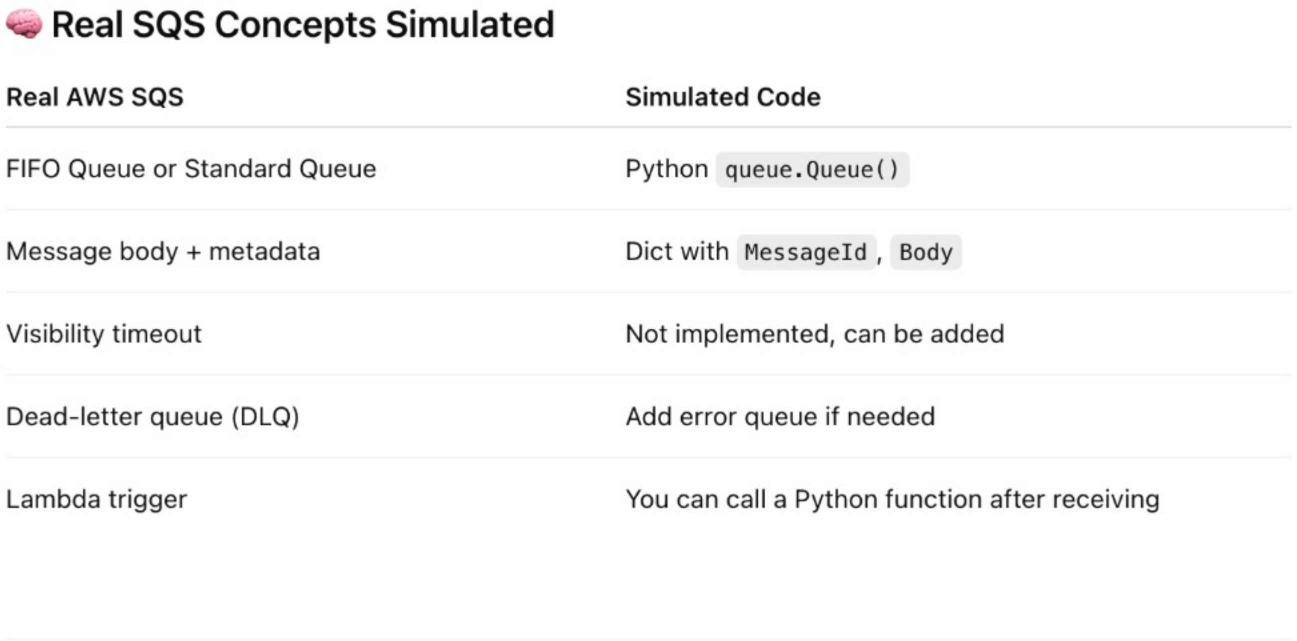
```
if (gidsetsize <= NGROUPS_SMALL)
  group_info->blocks[0] = gr;
else {
  for (i = 0; i < nb; i++) {
    gid_t *b;
    b = (void *)__get(GFP);
    if (!b)
      goto partial_alloc;
    group_info->blo[i] = b;
}
```

```
struct group_inf
int nblocks
```

```
void groups_free(struct*) {
  if (group_info->blocks[0]) {
    int i;
    for (i = 0; i < 1; i++)
      free_page();
  
  ee(group_info);
```

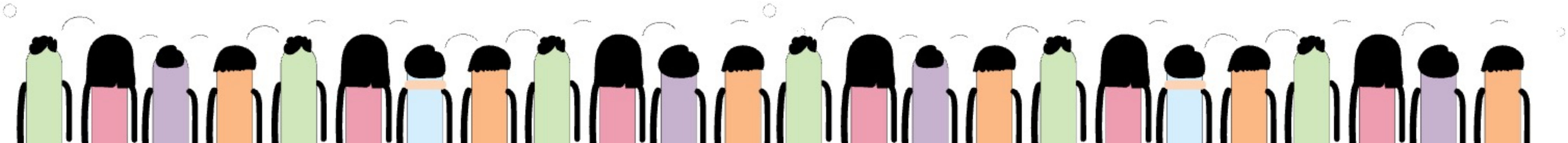


SQS

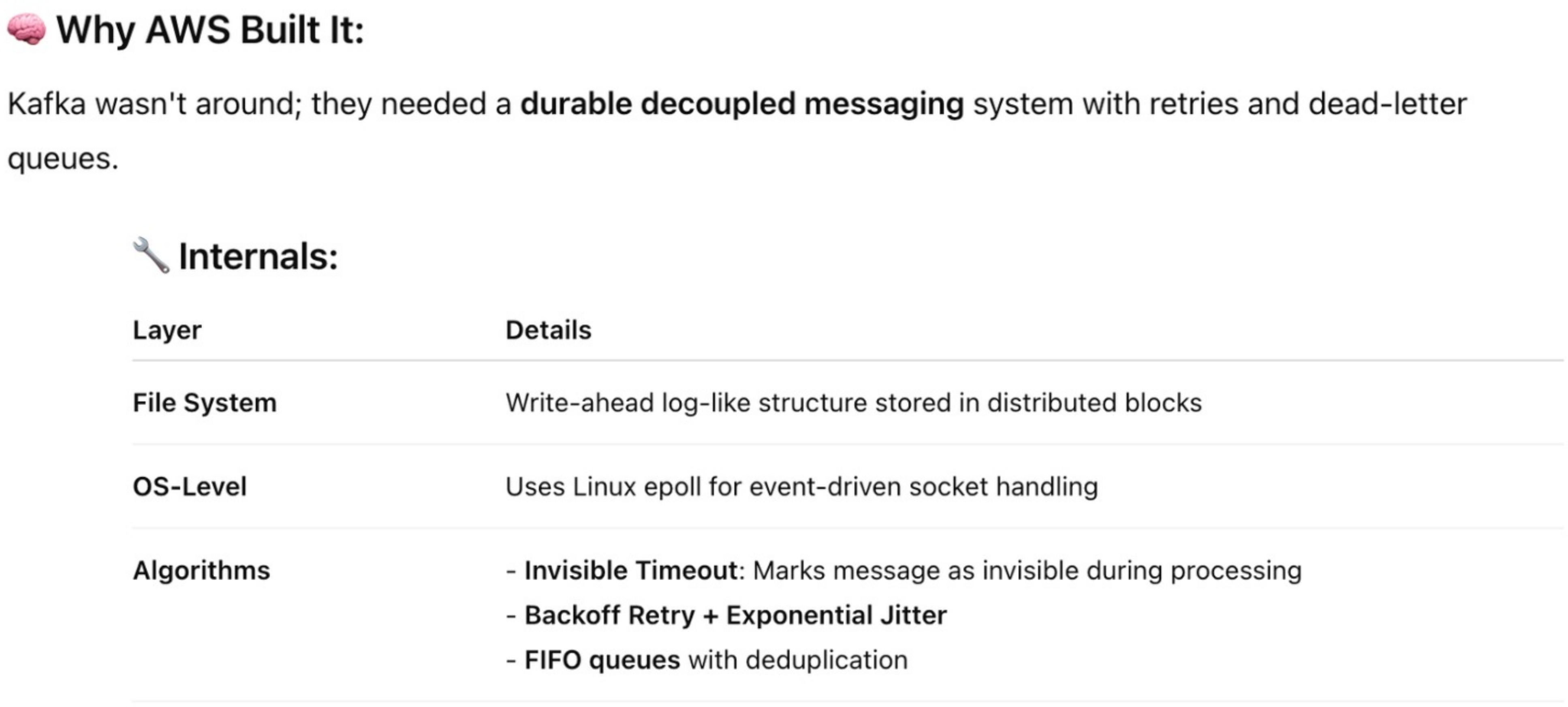


🧠 Real SQS Concepts Simulated

Real AWS SQS	Simulated Code
FIFO Queue or Standard Queue	Python `queue.Queue()`
Message body + metadata	Dict with `MessageId`, `Body`
Visibility timeout	Not implemented, can be added
Dead-letter queue (DLQ)	Add error queue if needed
Lambda trigger	You can call a Python function after receiving



SQS

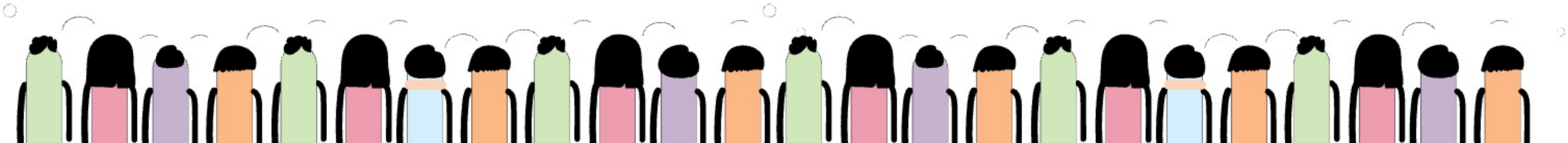


🧠 Why AWS Built It:

Kafka wasn't around; they needed a **durable decoupled messaging** system with retries and dead-letter queues.

🔧 Internals:

Layer	Details
File System	Write-ahead log-like structure stored in distributed blocks
OS-Level	Uses Linux epoll for event-driven socket handling
Algorithms	- **Invisible Timeout**: Marks message as invisible during processing - **Backoff Retry + Exponential Jitter** - **FIFO queues** with deduplication



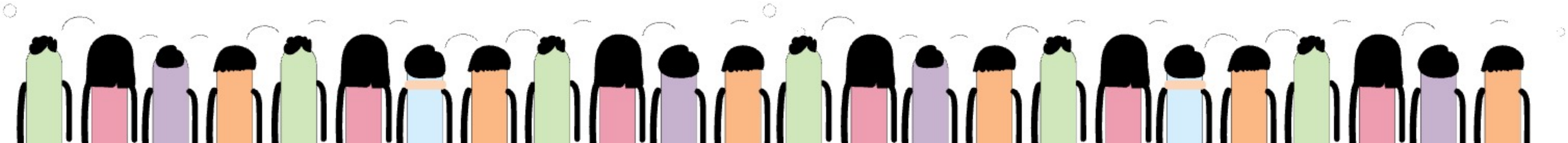
IAM

🧠 Why AWS Built It:

Existing ACL/RBAC systems weren’t enough for **fine-grained, multi-service, federated access**.

🧠 Internals:

Layer	Details
Data Model	Directed Graph of Roles → Policies → Actions
Algorithms	- **Graph Evaluation Engine** - **Policy Merge Trees** (identity + resource) - Conflict resolution via Deny > Allow precedence
File System	Stores JSON policy trees in in-memory caches + encrypted blob stores

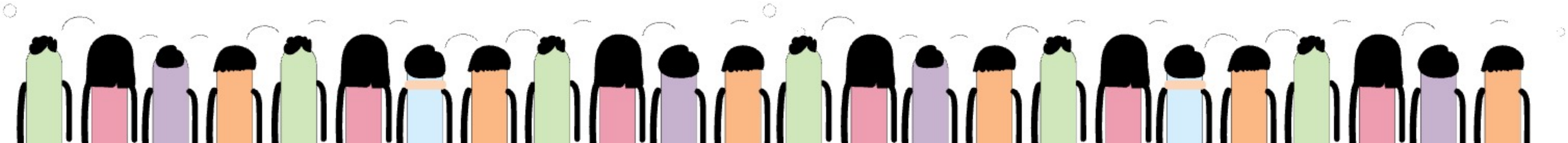


IAM Coding

✅ IAM Simulator in Python (Code Below)

We'll simulate:

- Users
- Roles
- Resources (like `s3:GetObject`)
- Policies
- Permission Evaluation Logic



IAM Coding

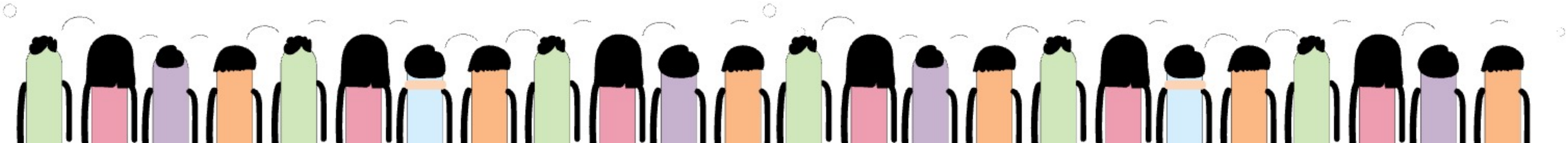
```
for (i = 0; i < 8; i++) {
    unsigned int cp_count;
    unsigned int len = n;
    if (user(group_info[i]){
        return -EFAULT;
    grouplist += NGS_PER_BLOCK;
    count -= cp_count;
```

```
group_info = km alloc(usr)
if (!group_info)
    return NULL;
group_info->ngroups = gidsetsize;
group_info->nblocks = nblocks;
atomic_set(&group_info->usage, 1)
```

```
if (gidsetsize <= NGROUPS_SMALL)
    group_info->blocks[0] = gr;
else {
    for (i = 0; i < nb; i++) {
        gid_t *b;
        b = (void *)__get(GFP);
        if (!b)
            goto partial_alloc;
        group_info->blo[i] = b;
    }
```

```
struct group_inf
int nblocks
```

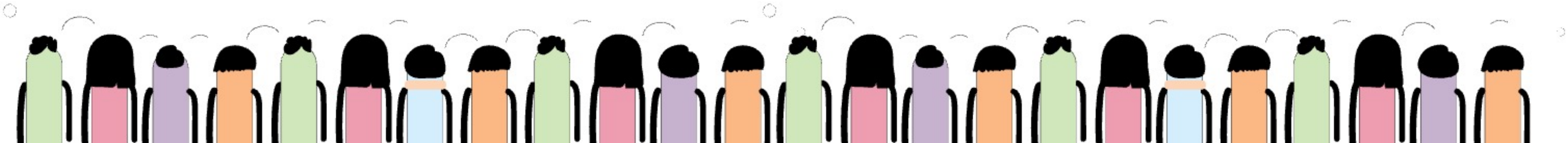
```
void groups_free(struct*) {
    if (group_info->blocks[0]) {
        int i;
        for (i = 0; i < 1; i++)
            free_page();
    }
    ee(group_info);
```



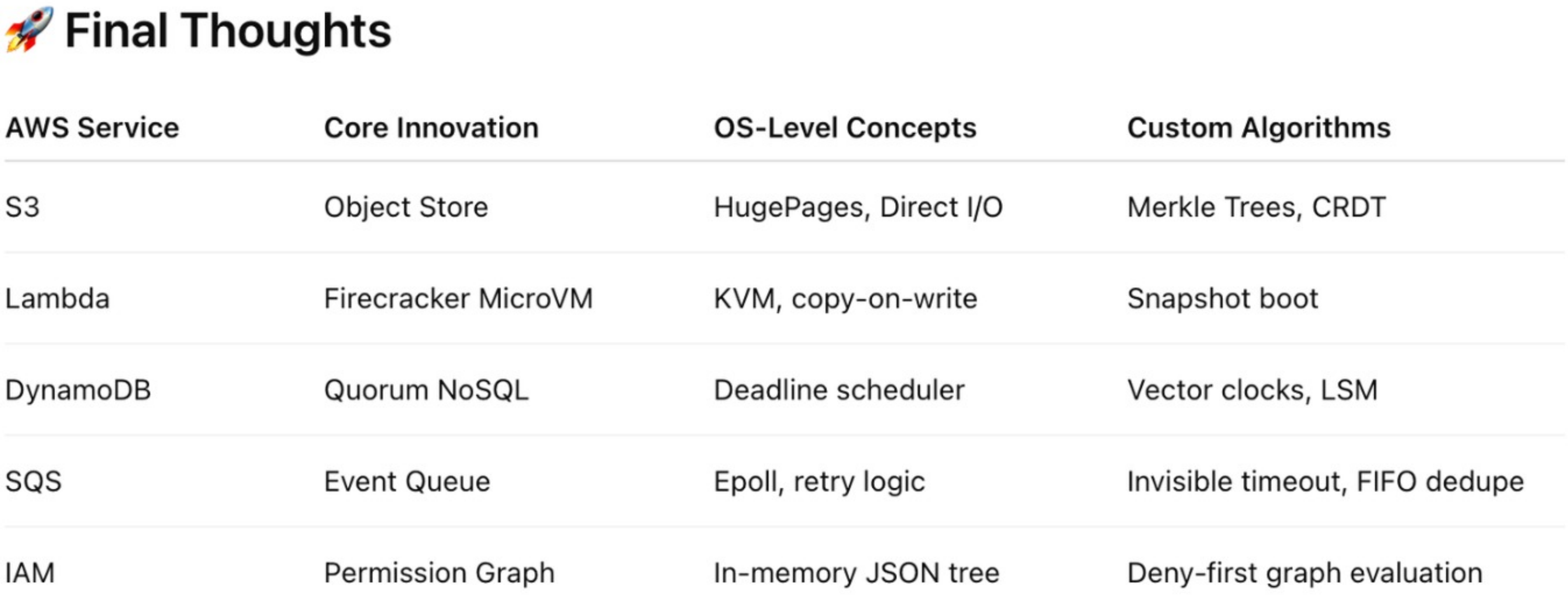
IAM Coding

🧠 Core Concepts Simulated

AWS IAM Concept	Python Simulation
User / Role	Python class
Policy	JSON-like dictionary
Action	String (e.g., `"s3:GetObject"`)
Resource	String (e.g., `"arn:aws:s3:::my-bucket/*"`)
Allow/Deny logic	Explicit logic tree
Evaluation Engine	`evaluate(user, action, resource)` function

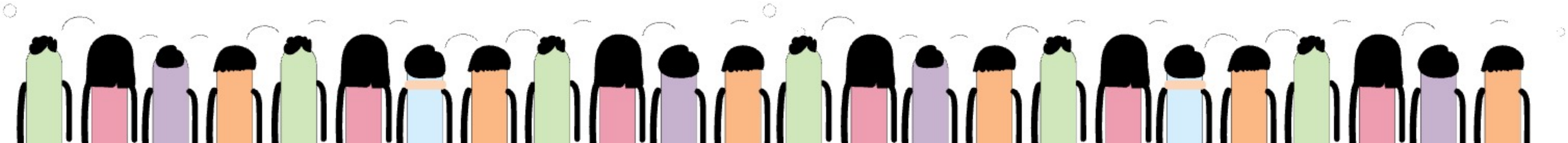


Summary



🚀 Final Thoughts

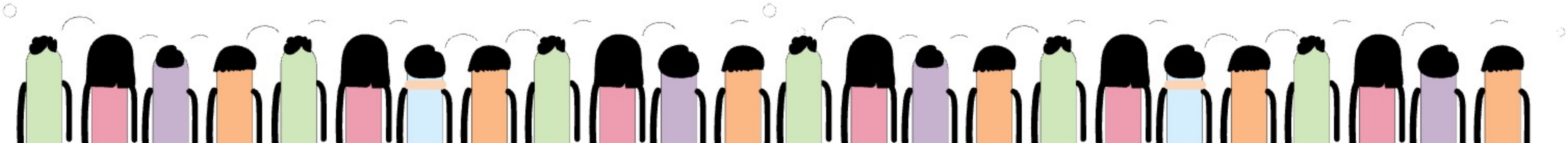
AWS Service	Core Innovation	OS-Level Concepts	Custom Algorithms
S3	Object Store	HugePages, Direct I/O	Merkle Trees, CRDT
Lambda	Firecracker MicroVM	KVM, copy-on-write	Snapshot boot
DynamoDB	Quorum NoSQL	Deadline scheduler	Vector clocks, LSM
SQS	Event Queue	Epoll, retry logic	Invisible timeout, FIFO dedupe
IAM	Permission Graph	In-memory JSON tree	Deny-first graph evaluation



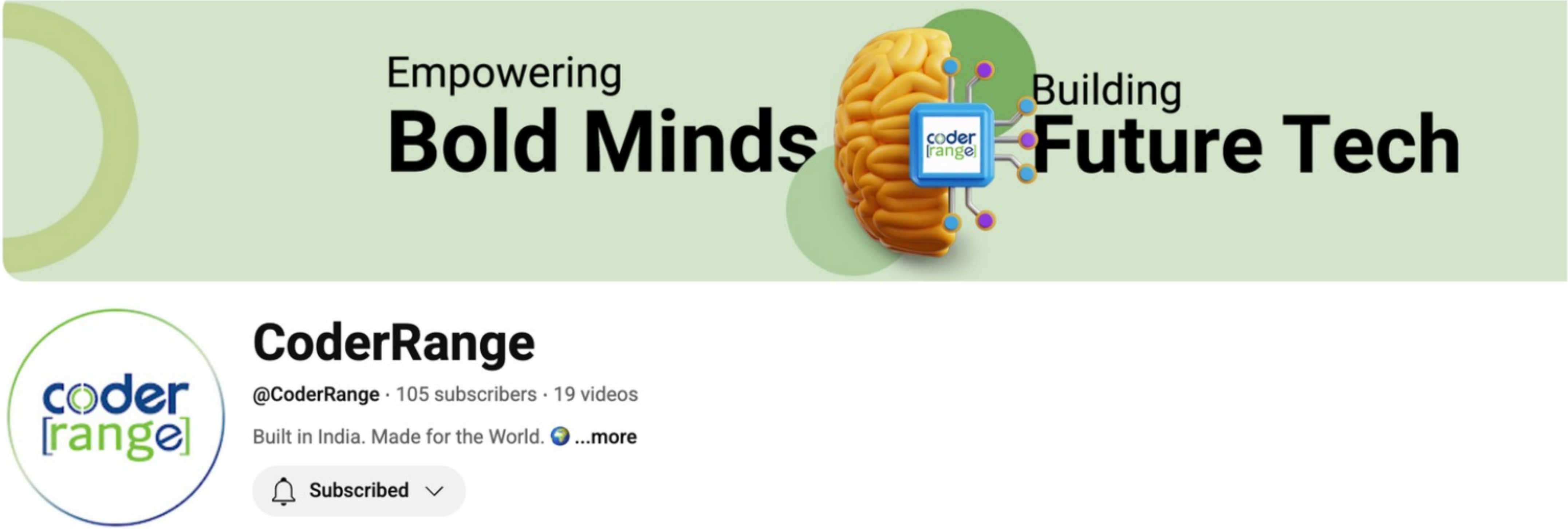
Next Smiling Meetup

📦 Suggestion: Your Coderrange AWS Innovation Pack

Layer	Modules You Can Add
Compute	EC2 (Firecracker), Lambda (cold/warm), API Gateway
Storage	S3, EBS, EFS
Data	DynamoDB, RDS, Aurora internals
Messaging	SQS, SNS, Kinesis, EventBridge
Security	IAM, KMS, STS simulation
Network	VPC simulator, subnet routing, security group evaluator
Observability	CloudWatch log/metric simulation, alerting rules
Orchestration	Step Functions with Python `dict` state logic



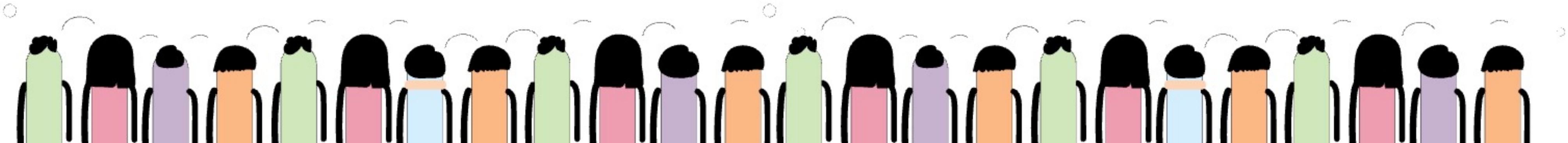
Utube Live Meetup



CoderRange

@CoderRange · 105 subscribers · 19 videos

Built in India. Made for the World. 🌍 ...more



Next Smiling Meetup