ALPS: Automated Least-Privilege Enforcement for Securing Serverless Functions

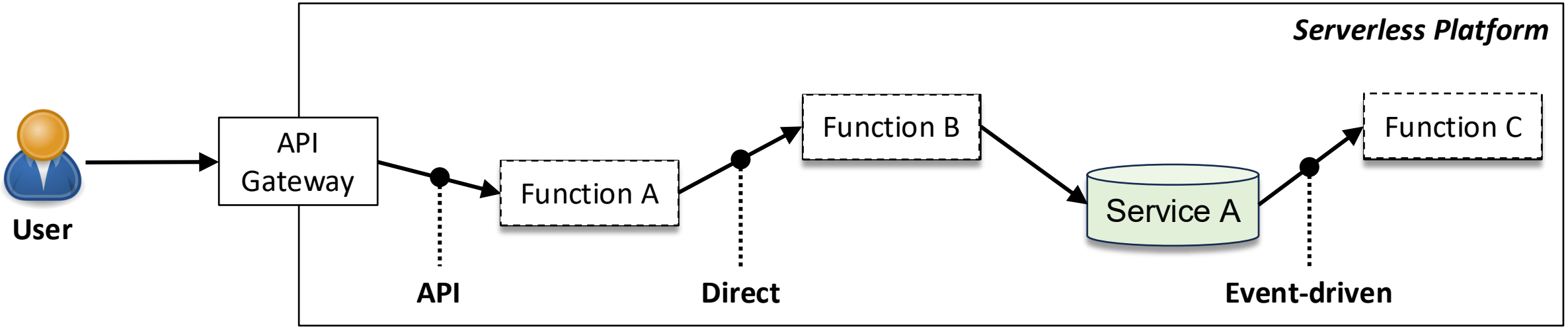
Changhee Shin*, Bom Kim**, and Seungsoo Lee*

* Department of Computer Science & Engineering
Incheon National University

** School of Electrical Engineering
KAIST

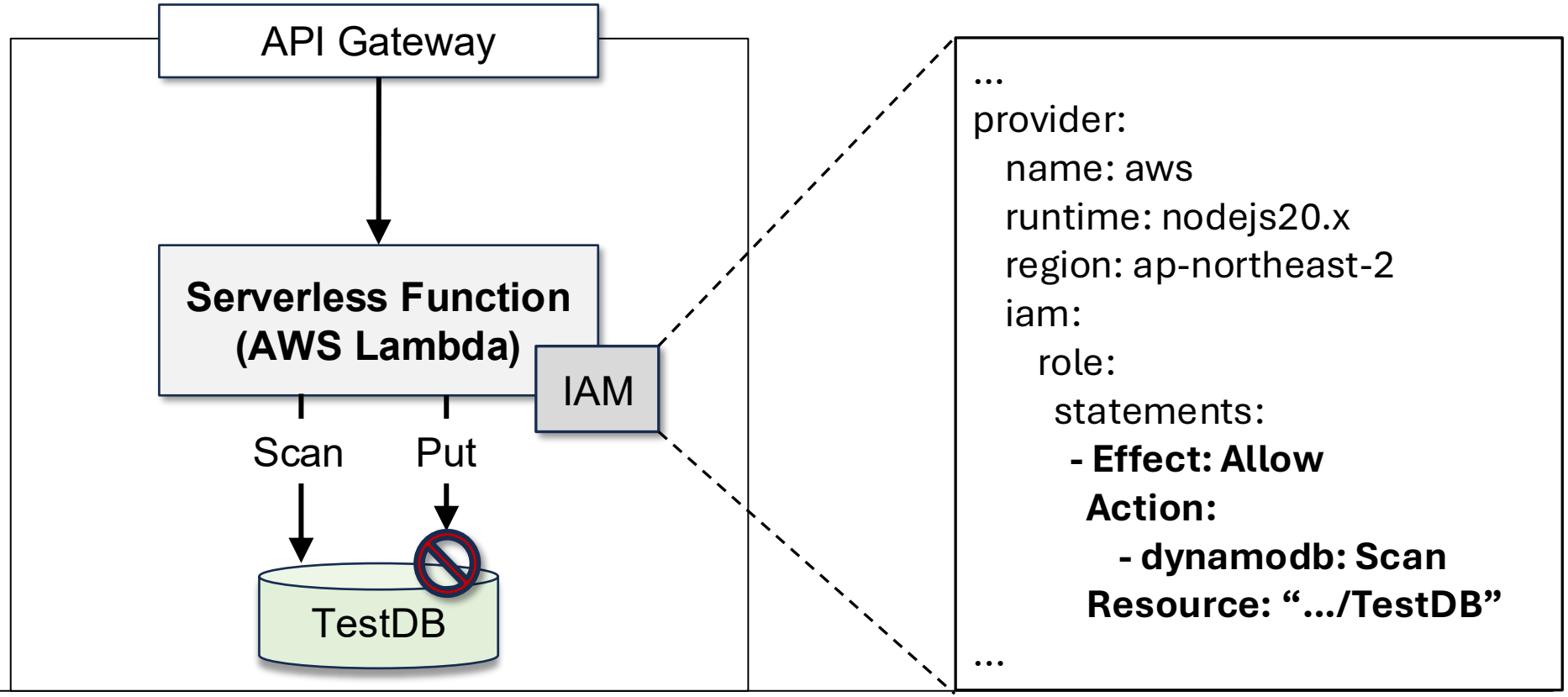
Serverless Function Invocation

- Stateless functions triggered through **three invocation methods**
  → API (HTTP), Direct (function-to-function), Event-driven (state change)



IAM-based Access Control

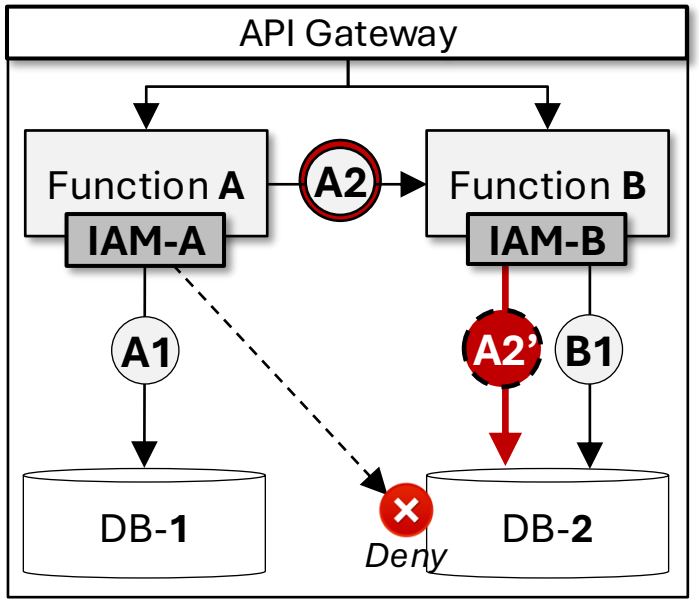
- IAM(Identity and Access Management)
  - Controls function-to-resource access by **evaluating policies on every invocation**

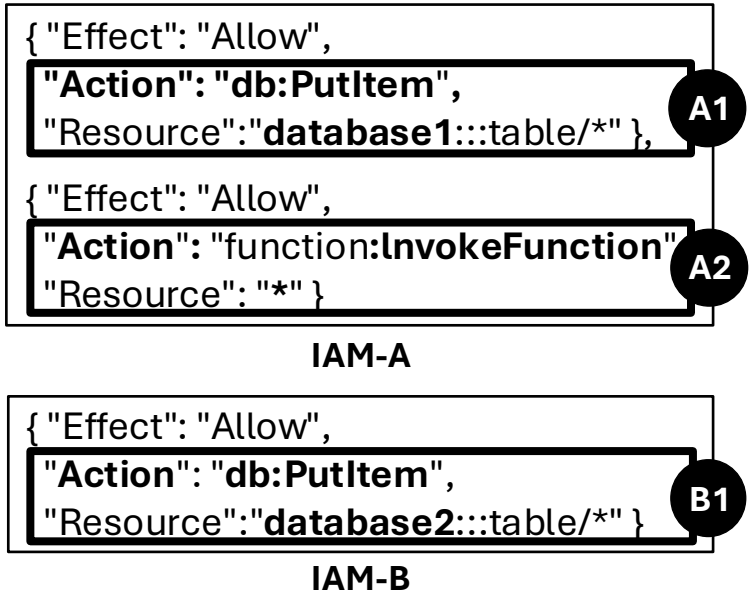


Problem Statements (1/3)

- Manual policy management
  → **over-privileged functions** and **indirect abuse**

< sysdig 2024 Cloud-Native Security and Usage Report >

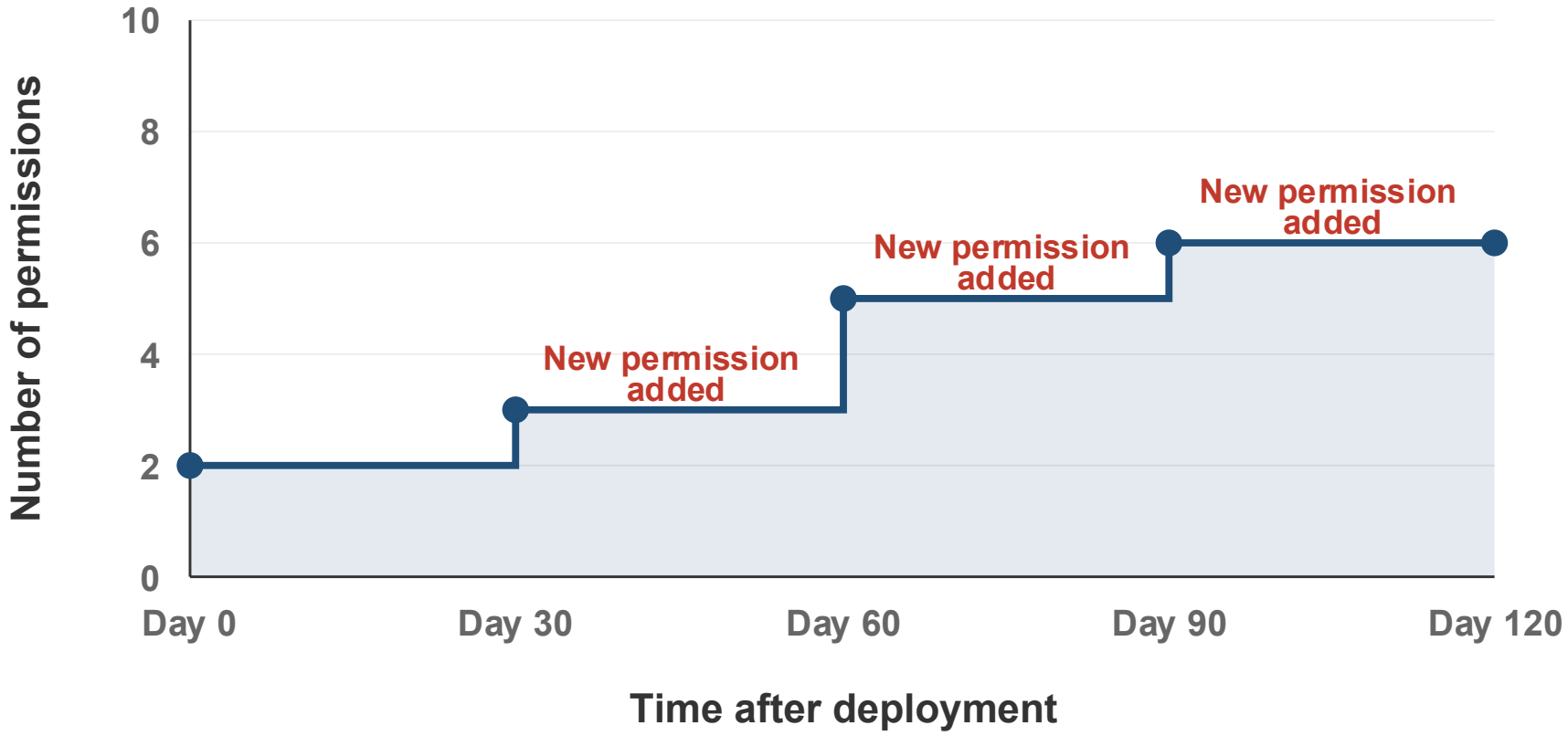




Sysdig report: https://2631050.fs1.hubspotusercontent-na1.net/hubfs/2631050/Sysdig%202024-report-cloud-native-security-and-usage.pdf

Problem Statements (2/3)

- **Privilege drift** post-deployment with **no runtime enforcement**



Set and Forget Problem: https://assets.barracuda.com/assets/docs/dms/checklist_Identity_and_access_management_ENG.pdf

Problem Statements (3/3)

- **Distinct IAM** grammars and SDK styles **across vendors and languages**

AWS IAM

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "s3:PutObject",
    "Resource": "arn:aws:s3:::my-bucket/*"
  }]
}
```

Google Cloud IAM

```
bindings:
- role: roles/storage.objectCreator
  members:
  - serviceAccount:func@project.iam
  condition:
    expression: resource.name == "my-bucket"
```

Azure IAM

```
{
  "Name": "Blob Writer",
  "Actions": [

"Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write"
  ],
  "AssignableScopes": [
    "/subscriptions/{sub-id}"
  ]
}
```

ALPS Design Considerations

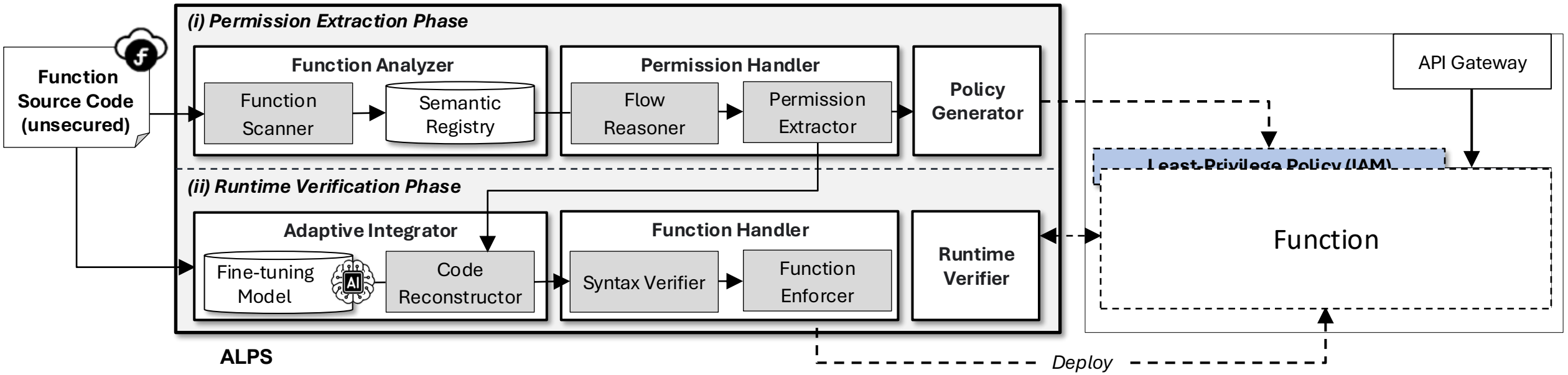
1. **Automated Permission Extraction**
   - **Static analysis** + **data flow inference** for fine-grained permissions

2. **Real-time Abuse Prevention**
   - Allowlist-based **pre-execution validation** of all SDK calls

3. **Cross-Vendor Code Integration**
   - **Context-aware code generation** via fine-tuned LLM

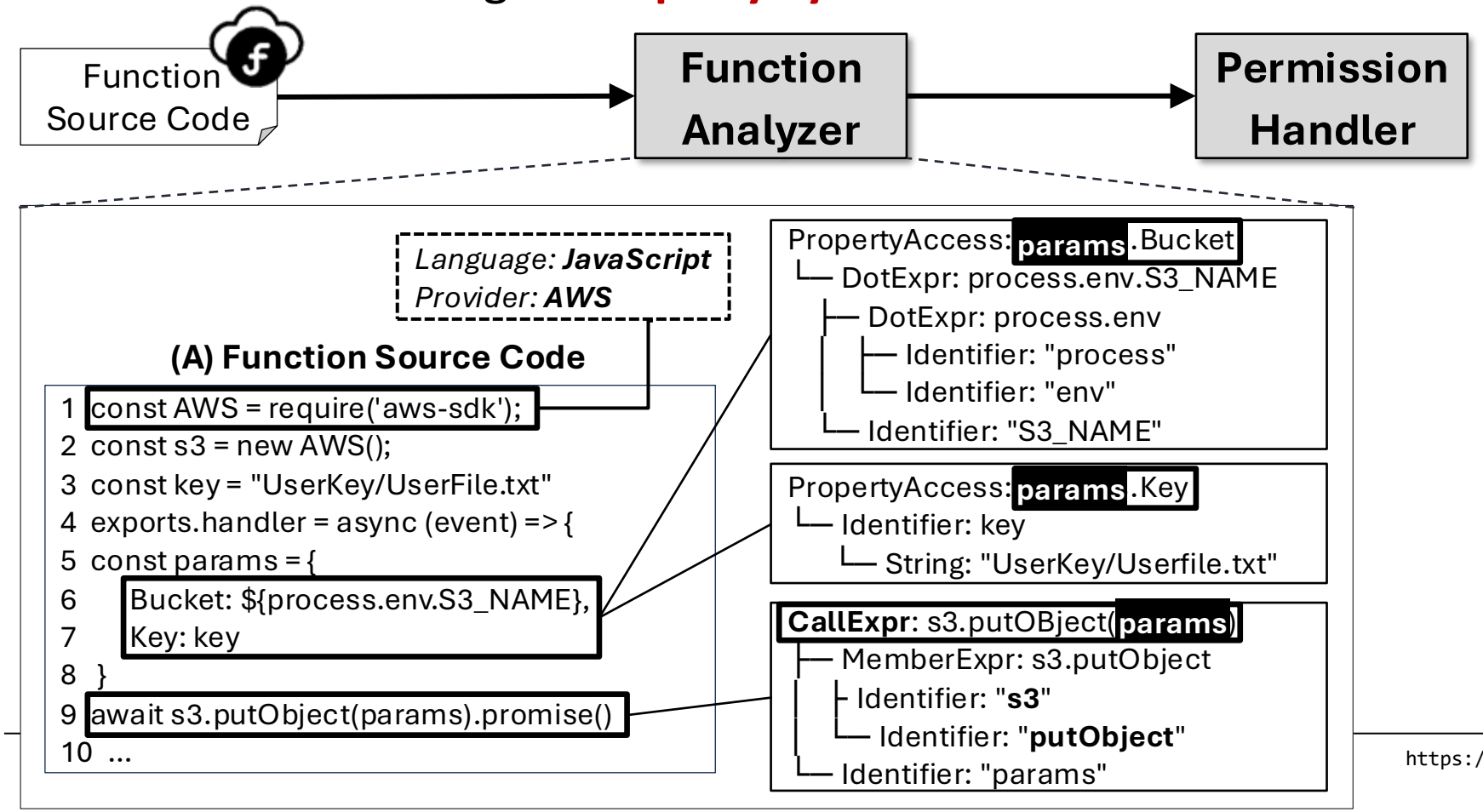
ALPS Overviews

1. Permission Extraction Phase
2. Runtime Verification Phase



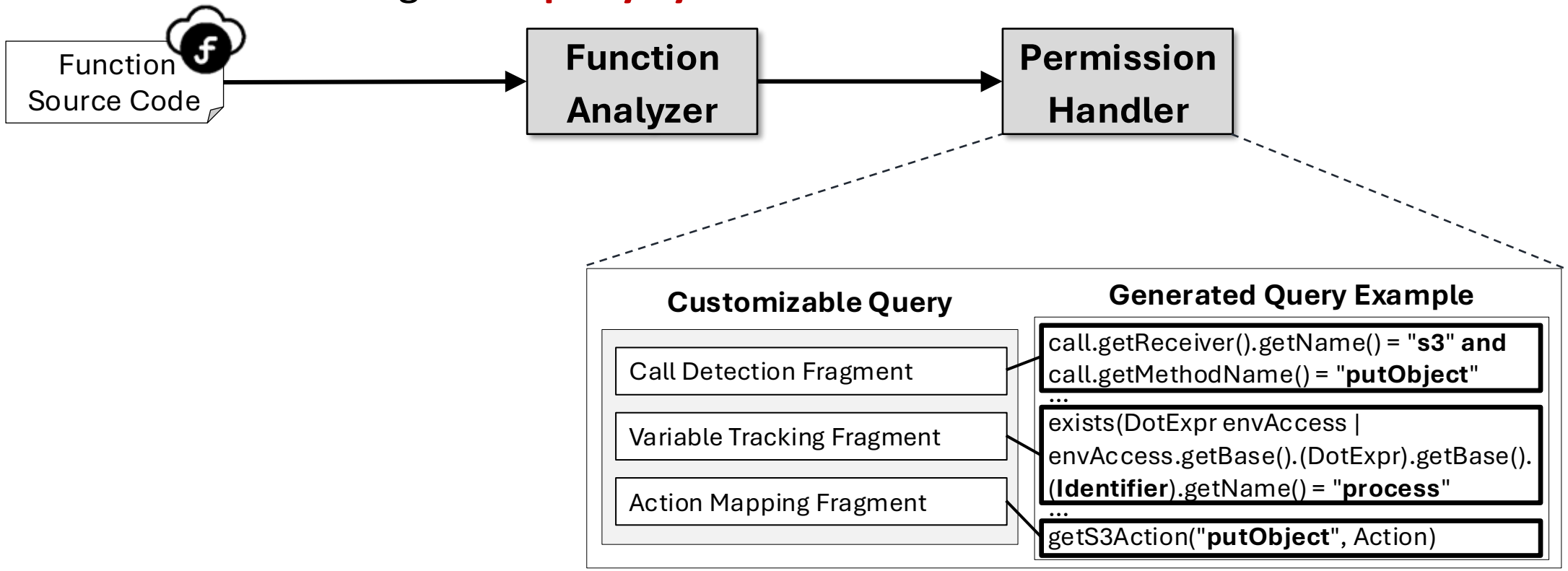
ALPS Details: Permission Extraction

- **Function Analyzer**: language/vendor detection + AST in Semantic Registry
- **Permission Handler**: 3-fragment **query system**



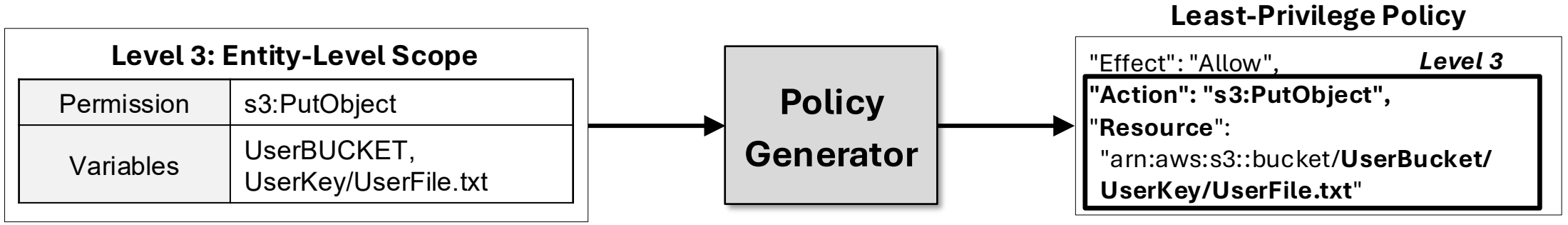
ALPS Details: Permission Extraction

- **Function Analyzer**: language/vendor detection + AST in Semantic Registry
- **Permission Handler**: 3-fragment **query system**



ALPS Details: Three-Level Granuarity

- **Policy Generator**: Generates vendor-specific policies with **three granularity levels**

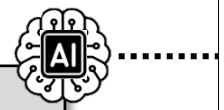


ALPS Details: Adaptive Code Integration

- **Adaptive Integrator**: Fine-tuned LLM injects **permission hooks** tailored to each language and vendor

(A) Initial Function Code (Google Cloud/Py)

```
def monitor_gcp(request):
    client = storage.Client()
    buckets = list(client.list_buckets())
    bucket_names = [b.name for b in buckets]
    return json.dumps({"buckets": bucket_names})
```



Adaptive Integrator

(B) Reconstructed Function Code

```
ALLOWED_CALLS = {
 f"storage:{os.getenv('BUCKET_NAME')}":["list"],"storage:buckets": ["list"] … }
class MonitoredClient(storage.Client):
 def list_buckets(self):
  resource,operation = "storage:buckets","list"
  if resource not in ALLOWED_CALLS or
     operation not in ALLOWED_CALLS[resource]:
       raise Exception(f"Unauthorized …")
       return super().list_buckets()
 def monitor_gcp(request): …
```

Instruction Guideline :

Here is a **{language} {vendor} function**, along with its **extracted permissions and variables**.

Based on the given information :
✓ **Insert permission hook code** to intercept all cloud service API calls.
✓ **Generate allowed call list** from extracted permissions and environment variables
✓ Return only the modified function code.

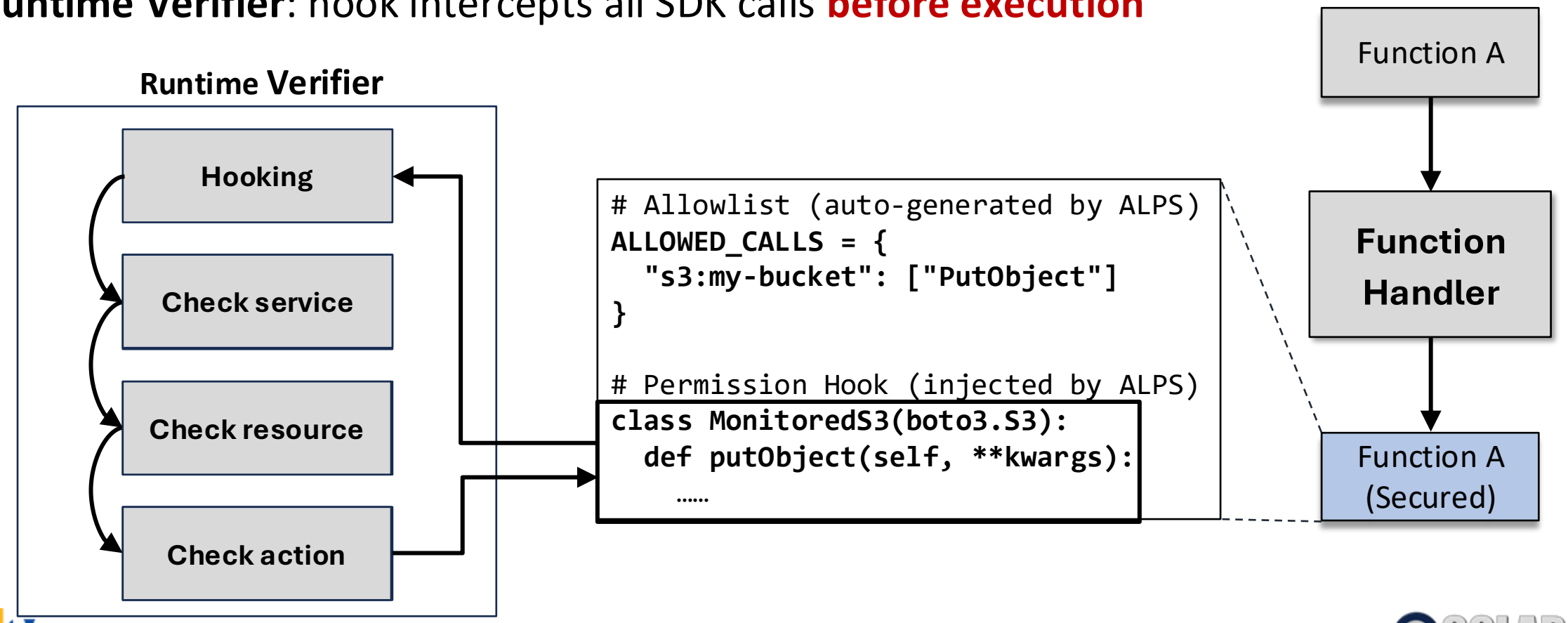
Input :

"code": ["< **original function source** >"],
"least_privilege":
["**<service>:<action>:<variable>**"]

Expected Output : Complete serverless function

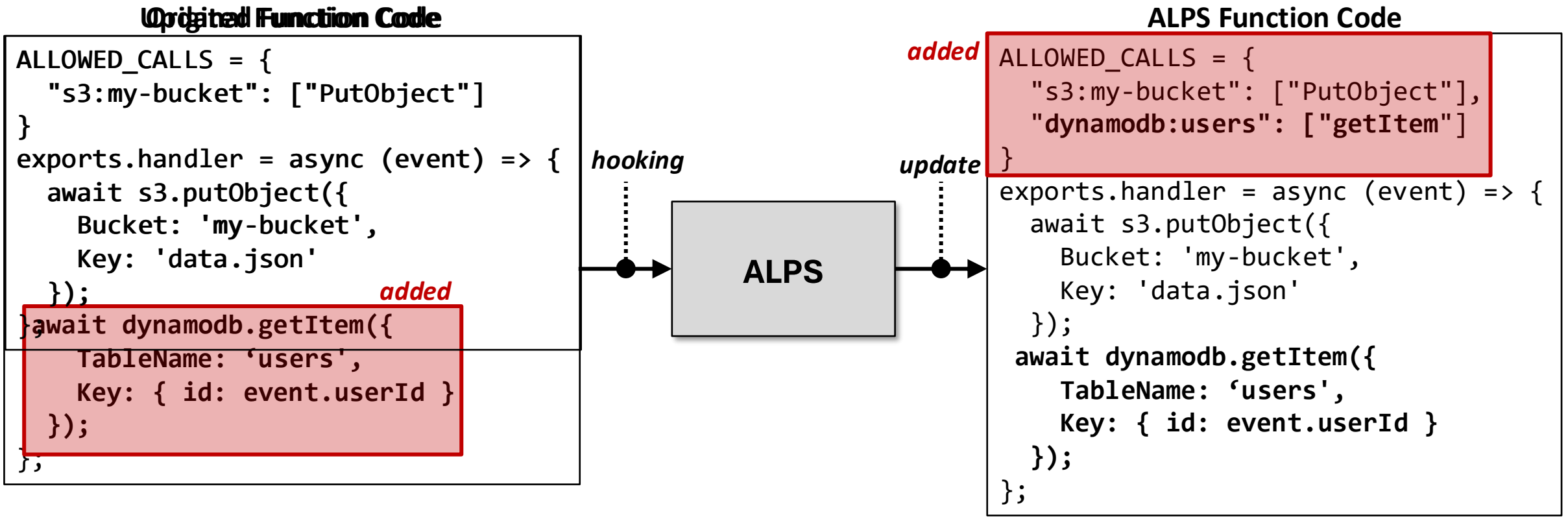
ALPS Details: Runtime Verification

- **Function Handler**: Syntax Verifier + Function Enforcer (validate before deploy)
- **Runtime Verifier**: hook intercepts all SDK calls **before execution**



ALPS Details: Post-Deployment Adaption

- Code change → Re-extract / Policy drift → Restrict to allowlist



Evaluation: Functional Correctness

- **Scenario**: Function A with excessive lambda:invoke → DB-B write attempt
- **Without ALPS**: malicious data inserted (**success**)
- **With ALPS**: **blocked** - Invalid AWS API Call: lambda:invoke

```
$ curl -X POST
https://malicious.app/api/v1/function-a -d
'{"user_id": "maliciousId", "password":
"root123!"}'
 {"status": 200, "body": {\"result\":
\"Success\"}
$ aws dynamodb get-item --table-name DB-B --
key '{"id": {"S": "maliciousId"}}'
 Item: {"user_id": {"S": "maliciousId"},
"password": {"S": "root123!" }}
```

(A) Attack execution before applying least-privilege policy

```
$ curl -X POST
https://malicious.app/api/v1/function-a -d
'{"user_id": "maliciousId", "password":
"root123!"}'
{"status": 400, "errorType": "Error",
 "errorMessage": "Unauthorized AWS API Call:
lambda:function-B.invoke"}
$ aws dynamodb get-item --table-name DB-B --
key '{"id": {"S": "maliciousId"}}'
 Item {}
```

(B) Attack execution after applying least privilege policy

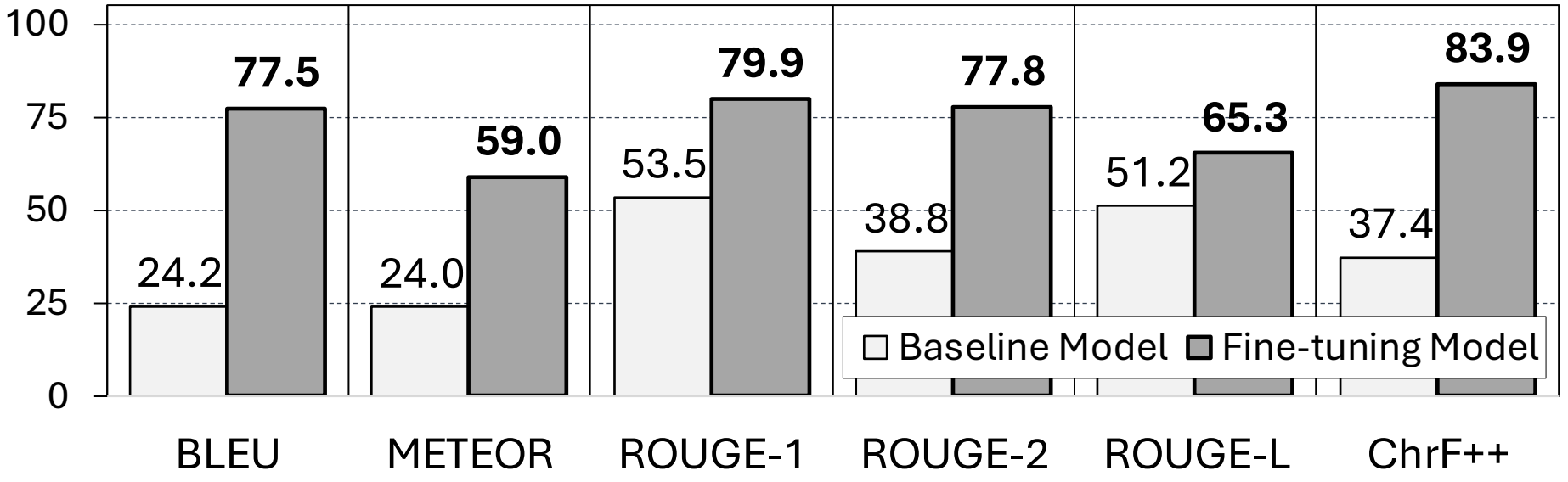
Evaluation: Function Coverage

- Achieves **94.8%** entity-level coverage on 8,322 functions across AWS / Google Cloud / Azure × JS / Python / Go
- Reaches the highest success rate on Google Cloud (**96.8%**), followed by AWS (**95.2%**) and Azure (**90.3%**)
- Falls back to service-level scope for the remaining **5.2%**

	AWS			Google Cloud			Azure		Total
	JS	Py	Go	JS	Py	Go	JS	Py	
Function	2512	1208	807	784	802	753	852	604	8322
Detected	2393	1140	779	759	777	729	827	587	7891
Undetected	119	68	28	25	25	24	25	17	331

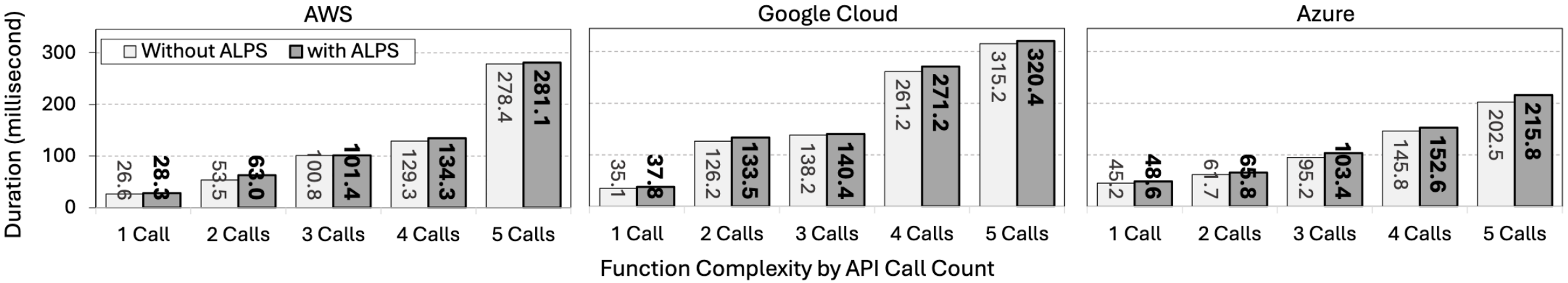
Evaluation: Fine-tuning LLM Performance

- Achieves a **220%** increase in BLEU after fine-tuning (**24.2 → 77.5**)
- Achieves a **100%** increase in ROUGE-2 after fine-tuning (**38.8 → 77.8**)
- Improves ChrF++ by **124%** (**37.4 → 83.9**), with consistent gains in METEOR and ROUGE-1/L



Evaluation: Performance Overhead

- Adds average execution overhead of **3.9ms** (AWS), **5.5ms** (Google Cloud), **7.2ms** (Azure)



Conclusion

- ***Automated*** Framework for Least-Privilege Enforcement
  - ALPS provides a fully automated, vendor-agnostic framework for enforcing least privilege in real-world serverless environments.

- Static Analysis + Fine-tuned LLM ***Integration***
  - This is the first work to combine serverless-tailored static analysis with fine-tuned LLM-based code integration for runtime enforcement.

- ***Effective*** Defense with ***Practical*** Overhead
  - Effectively blocks privilege abuse at runtime while maintaining minimal performance overhead across major cloud providers.

THANK YOU

Q & A