



RAINBOW

Polaris Framework – Creating and Enforcing Complex SLOs

Thomas Pusztai

Distributed Systems Group, TU Wien, Austria

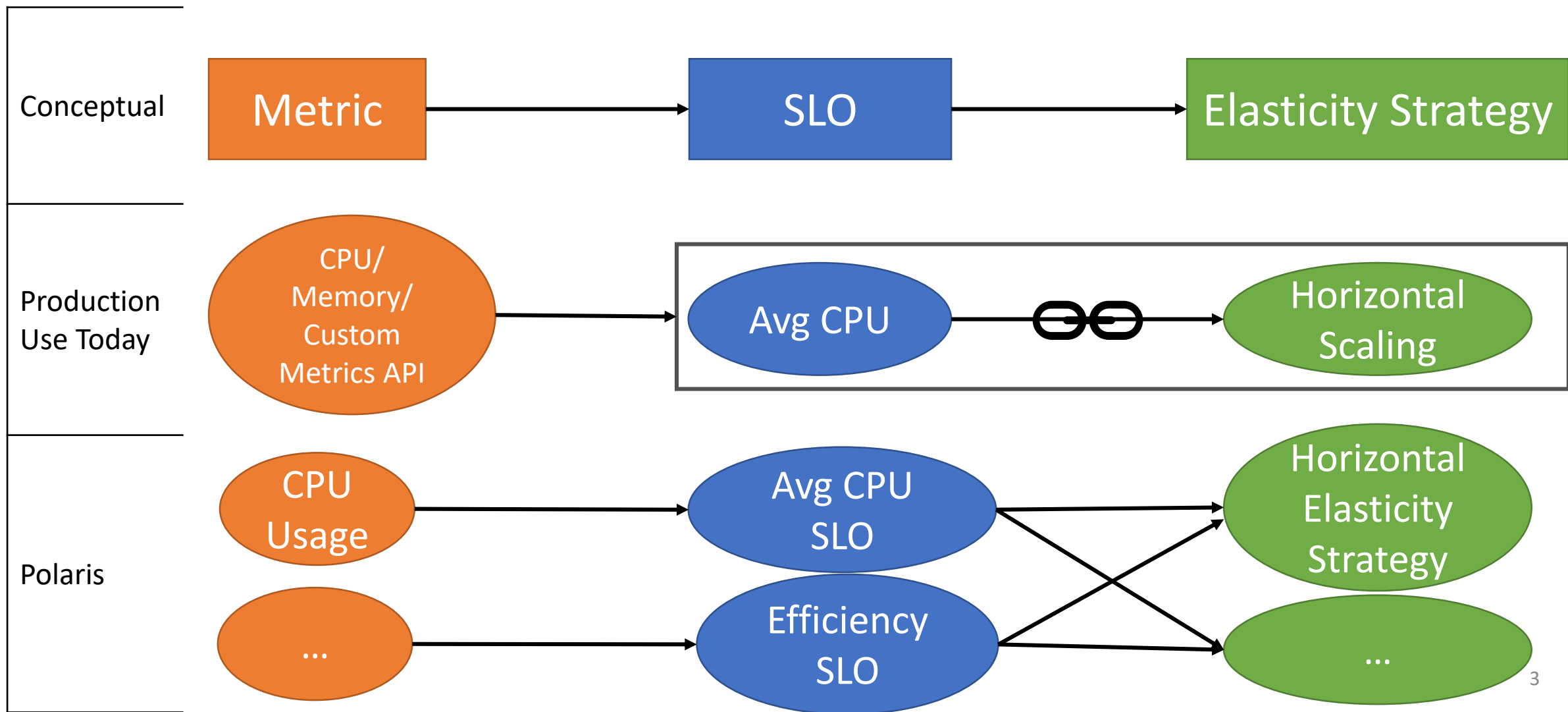
June 21, 2022

SLAs & SLOs

- Service Level Agreements (SLAs) define bounds within which a service has to operate
- An SLA consists of one or more Service Level Objectives (SLOs)
- SLO is a “commitment to maintain a particular state of the service in a given period” [1]
- SLOs are measurable
- Often limited to simple capacity guarantees, e.g., CPU, response time
- High-level SLOs (e.g., efficiency) would be suitable as business KPIs

[1] A. Keller and H. Ludwig. 2003. *The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web*. Journal of Network and Systems Management

SLO Components



Polaris

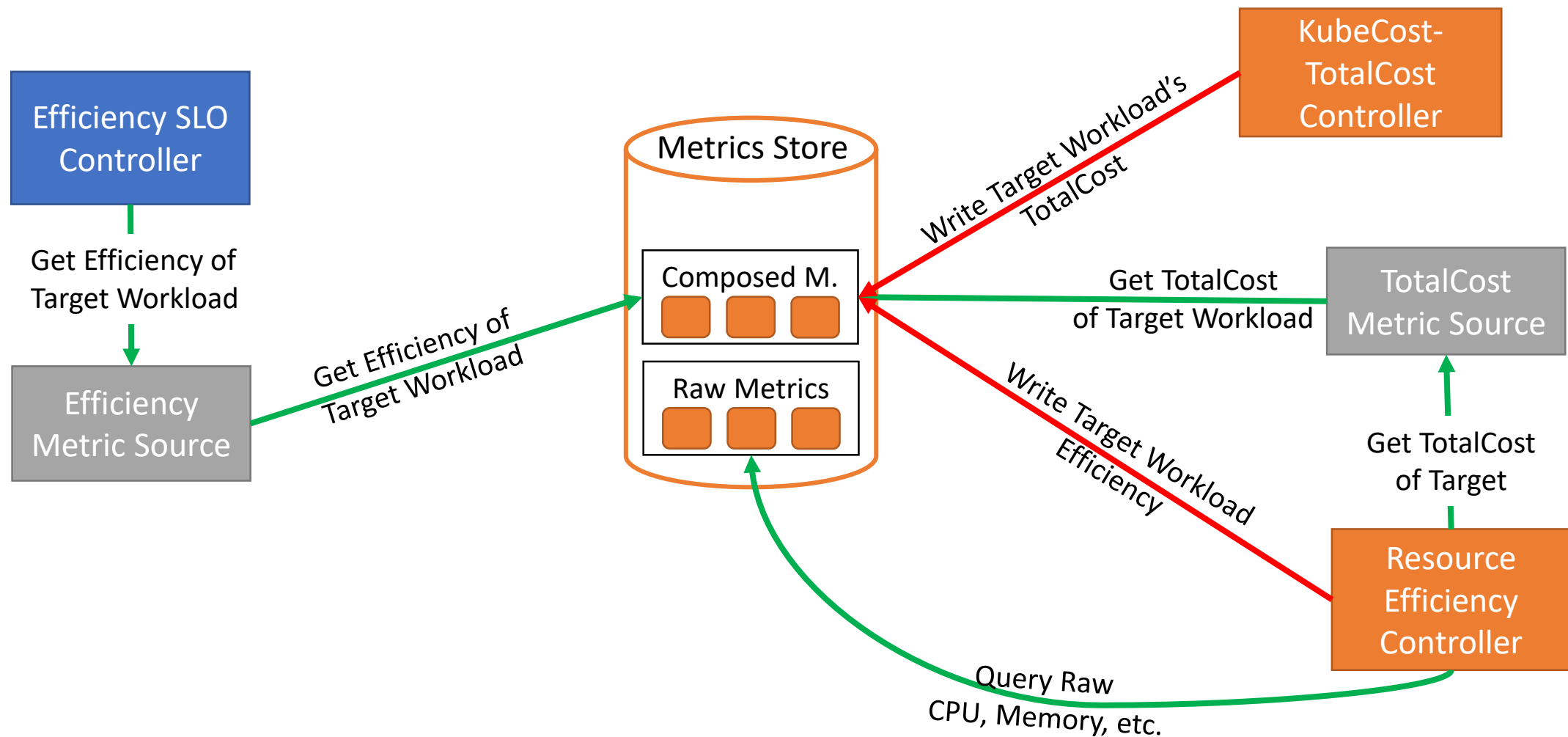


- TypeScript-based framework for building and customizing all three components
 - <https://polaris-slo-cloud.github.io>
- Decoupling of SLOs from elasticity strategies
- Increase the number of possible SLO/elasticity strategy combinations
- Available standalone, integrated into the RAINBOW platform, and (soon) integrated into the Centaurus platform

Polaris Composed Metrics

- High-level SLOs need high-level metrics
 - Not observable directly on system
 - Composed Metric = aggregation of multiple lower-level metrics
 - Often computed by an external controller and shared through a Metrics Store
 - Composed Metric can also be a prediction of future values of another metric

Resource Efficiency Composed Metric



Predicted metrics controllers

- Adds support for proactive scaling
- Polyglot approach using TypeScript and Python
- Implementation focuses on strengths of developers:
 - Polaris users
 - AI developers

Polaris SLOs

- SLO Controller periodically evaluates input metrics to check if the SLO is fulfilled or violated
- Triggers a user-configured elasticity strategy upon violation
- Easily buildable
 - Evaluation loop & triggering of elasticity strategy is implemented by Polaris
 - Only SLO business logic is required

```
export class EfficiencySlo
  implements ServiceLevelObjective<EfficiencySloConfig, SloCompliance>
{
  sloMapping: SloMapping<EfficiencySloConfig, SloCompliance>;

  private metricsSource: MetricsSource;
  private effMetricSource: ComposedMetricSource<Efficiency>;

  configure(
    sloMapping: SloMapping<EfficiencySloConfig, SloCompliance>,
    metricsSource: MetricsSource,
    orchestrator: OrchestratorGateway
  ): ObservableOrPromise<void> {
    this.sloMapping = sloMapping;
    this.metricsSource = metricsSource;

    const effMetricParams: EfficiencyParams = {
      namespace: sloMapping.metadata.namespace,
      sloTarget: sloMapping.spec.targetRef,
      owner: createOwnerReference(sloMapping),
    };
    this.effMetricSource = metricsSource.getComposedMetricSource(EfficiencyMetric.instance, effMetricParams);

    return of(undefined);
  }

  evaluate(): ObservableOrPromise<SloOutput<SloCompliance>> {
    return this.calculateSloCompliance()
      .then(sloCompliance => ({
        sloMapping: this.sloMapping,
        elasticityStrategyParams: {
          currSloCompliancePercentage: sloCompliance,
        },
      }));
  }

  private async calculateSloCompliance(): Promise<number> {
    const eff = await this.effMetricSource.getCurrentValue().toPromise();
    if (!eff) {
      Logger.log('Obtaining efficiency metric returned:', eff);
      return 100;
    }
    const compliance = (this.sloMapping.spec.sloConfig.targetEfficiency / eff.value.efficiency)
    return Math.ceil(compliance);
  }
}
```


Polaris Elasticity Strategies

- Modify a workload's resources, instances, and/or configuration in response to an SLO violation
- Out of the box
 - Horizontal scaling
 - Vertical scaling
 - Migrate to another node
- Easily buildable
 - Custom elasticity strategies for a specific workload

```
export class HorizontalElasticityStrategyController extends HorizontalElasticityStrategyControllerBase<
  SloTarget,
  HorizontalElasticityStrategyConfig
> {

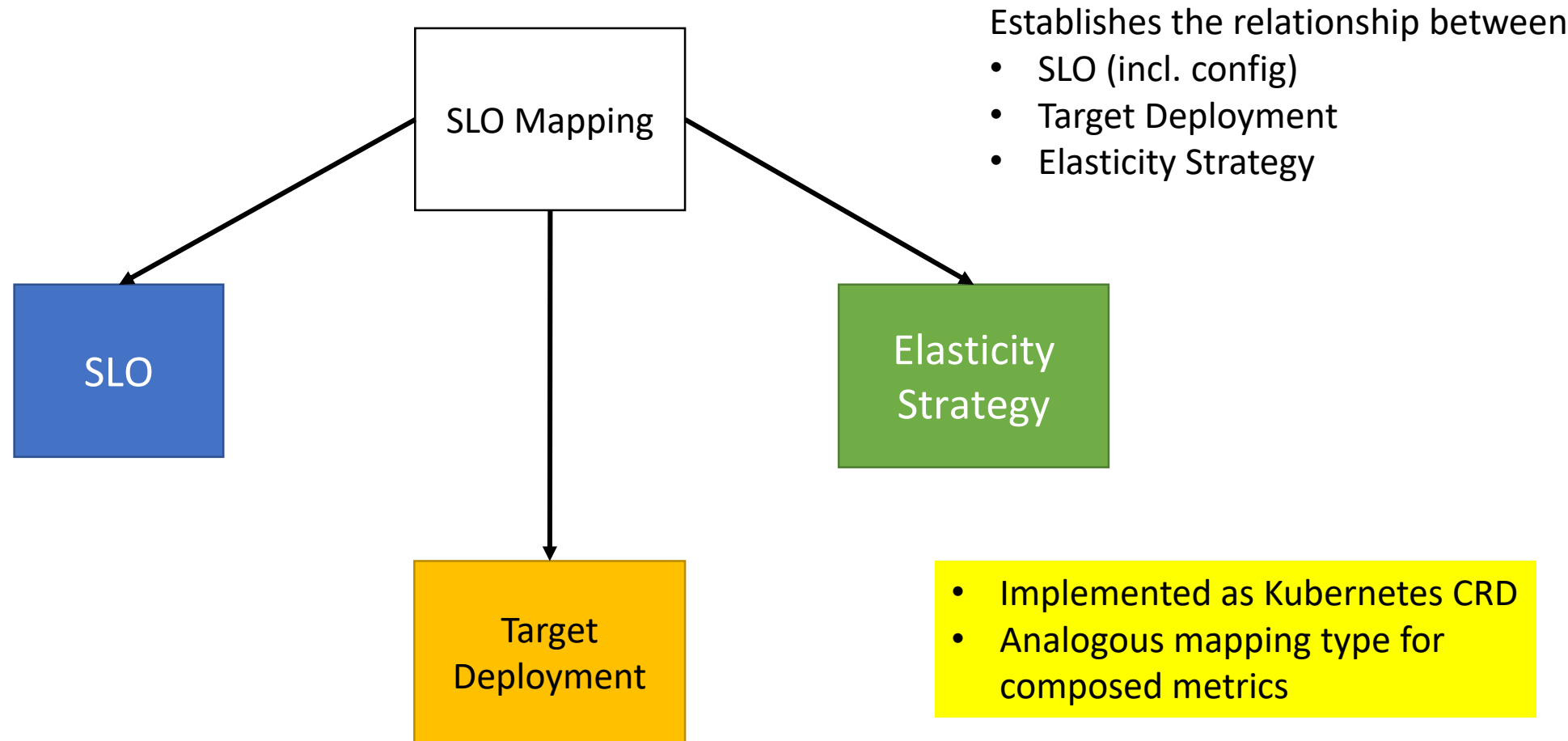
  protected computeScale(elasticityStrategy: ElasticityStrategy<SloCompliance, SloTarget, HorizontalElasticityStrategyConfig>): Promise<HorizontalElasticityStrategyConfig> {
    const newScale = new Scale(currScale);
    if (elasticityStrategy.spec.sloOutputParams.currSloCompliancePercentage > 100) {
      newScale.spec.replicas++;
    } else {
      newScale.spec.replicas--;
    }
    return Promise.resolve(newScale);
  }
}
```

```

stabilizationWindow: StabilizationWindow { scaleDownSeconds: 60, scaleUpSeconds: 40 },
staticConfig: { maxReplicas: 10, minReplicas: 1 },
targetRef: SloTarget {
  kind: 'Deployment',
  name: 'twitter-clone',
  group: 'apps',
  version: 'v1'
}
}
}
}
Successfully updated scale subresource from 1 to 2 replicas for: HorizontalElasticityStrategy
objectKind: ObjectKind {
  kind: 'HorizontalElasticityStrategy',
  group: 'elasticity.polaris-slo-cloud.github.io',
  version: 'v1'
},
metadata: ApiObjectMetadata {
  annotations: {
    'kubectl.kubernetes.io/last-applied-configuration': '{"apiVersion":"elasticity.polaris-slo-cloud.github.io/v1","kind":"HorizontalElasticityStrategy","metadata":{"annotations":{},"name":"horizontalelasticstrategy-sample","namespace":"default"},"spec":{"sloOutputParams":{"currSloCompliancePercentage":200,"tolerance":10},"stabilizationWindow":{"scaleDownSeconds":60,"scaleUpSeconds":40},"staticConfig":{"maxReplicas":10,"minReplicas":1},"targetRef":{"apiVersion":"apps/v1","kind":"Deployment","name":"twitter-clone"}}}'\n'
  },
  creationTimestamp: 2021-07-10T21:18:15.000Z,
  generation: 1,
  managedFields: [ [Object] ],
  name: 'horizontalelasticstrategy-sample',
  namespace: 'default',
  resourceVersion: '60782',
  uid: 'edd77639-5347-4477-8c40-88ecfd8f418c'
},
spec: ElasticityStrategySpec {
  sloOutputParams: { currSloCompliancePercentage: 200, tolerance: 10 },
  stabilizationWindow: StabilizationWindow { scaleDownSeconds: 60, scaleUpSeconds: 40 },
  staticConfig: { maxReplicas: 10, minReplicas: 1 },
  targetRef: SloTarget {
    kind: 'Deployment',
    name: 'twitter-clone',
    group: 'apps',
    version: 'v1'
  }
}
}
}

```

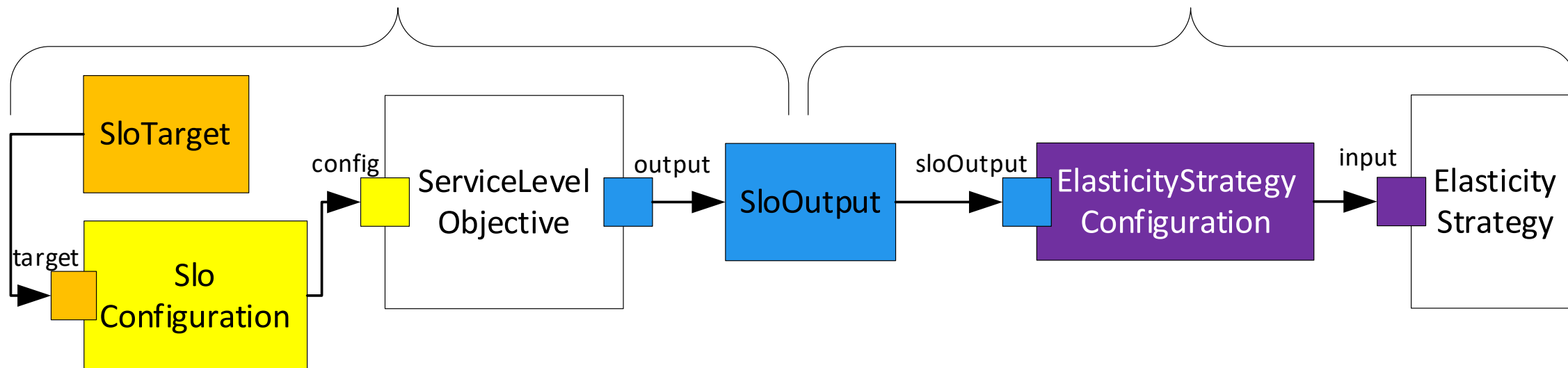
SLO Mapping



Type Safety

Types determined by ServiceLevelObjective

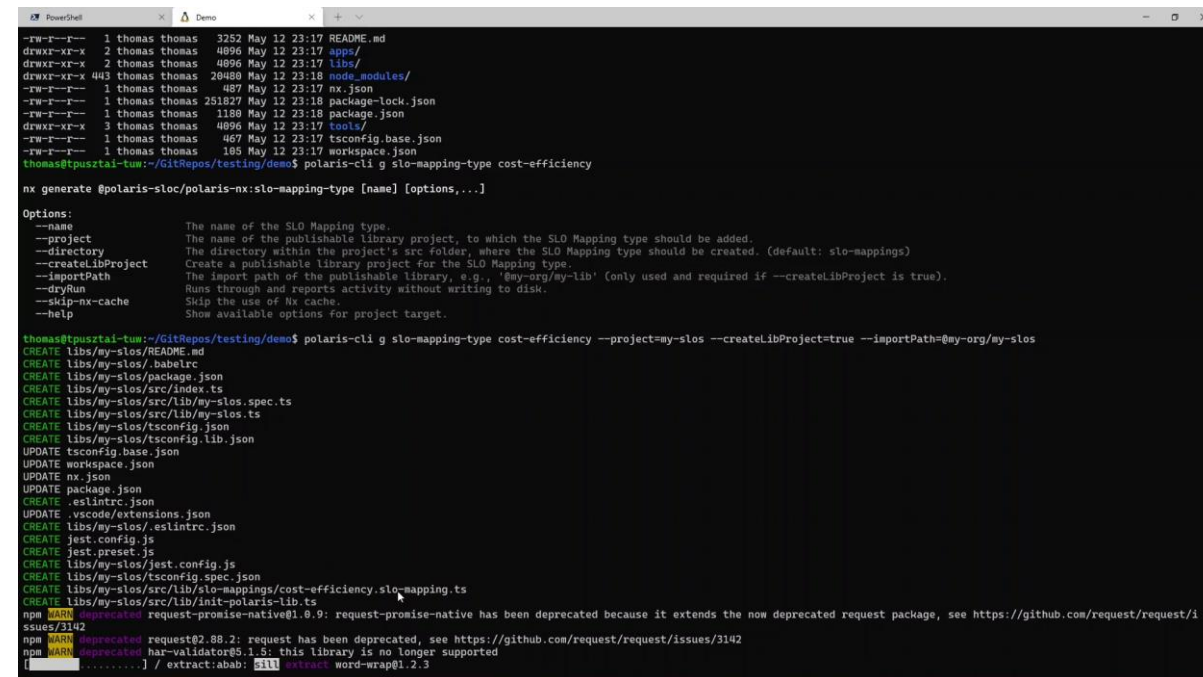
Types determined by ElasticityStrategy



Polaris as an Ecosystem

CLI and Development Containers

- A tool for service providers, developers and consumers:
 - Automatic setup of projects and dependencies
 - Create Composed Metric Types & Controllers
 - Create SLO Mappings & Controllers
 - Create Elasticity Strategy types & Controllers
 - Use and configure existing SLOs
- Lower entrance barrier for new developers
 - DevRel-first approach
 - Easy onboarding => more adopters
- Allows combining Polaris-based components with other projects/components in a monorepo
- RAINBOW users can leverage Web UI to configure existing SLOs



```
PowerShell Demo
1 thomas thomas 3252 May 12 23:17 README.md
drwxr-xr-x 2 thomas thomas 4096 May 12 23:17 apps/
drwxr-xr-x 2 thomas thomas 4096 May 12 23:17 libs/
drwxr-xr-x 443 thomas thomas 20480 May 12 23:18 node_modules/
-rw-r--r-- 1 thomas thomas 487 May 12 23:17 nx.json
-rw-r--r-- 1 thomas thomas 251827 May 12 23:18 package-lock.json
-rw-r--r-- 1 thomas thomas 1180 May 12 23:18 package.json
drwxr-xr-x 3 thomas thomas 4096 May 12 23:17 tools/
-rw-r--r-- 1 thomas thomas 467 May 12 23:17 tsconfig.base.json
-rw-r--r-- 1 thomas thomas 188 May 12 23:17 workspace.json
thomas@pusztai-tuw:~/GitRepos/testing/demo$ polaris-cli g slo-mapping-type cost-efficiency

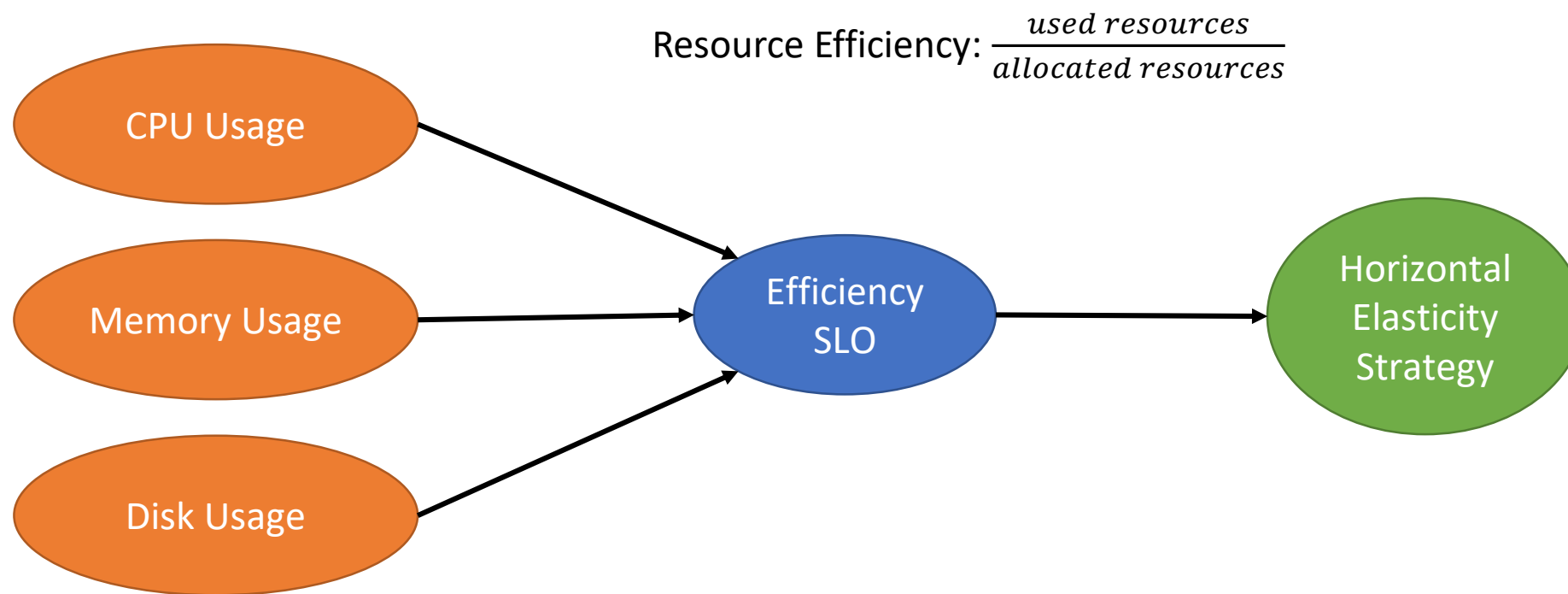
nx generate @polaris-sloc/polaris-nx:slo-mapping-type [name] [options,...]

Options:
  --name          The name of the SLO Mapping type.
  --project       The name of the publishable library project, to which the SLO Mapping type should be added.
  --directory     The directory within the project's src folder, where the SLO Mapping type should be created. (default: slo-mappings)
  --createLibProject Create a publishable library project for the SLO Mapping type.
  --importPath    The import path of the publishable library, e.g., '@my-org/my-lib' (only used and required if --createLibProject is true).
  --dryRun       Runs through and reports activity without writing to disk.
  --skipNxCache  Skip the use of Nx cache.
  --help         Show available options for project target.

thomas@pusztai-tuw:~/GitRepos/testing/demo$ polaris-cli g slo-mapping-type cost-efficiency --project=my-slos --createLibProject=true --importPath=@my-org/my-slos
CREATE libs/my-slos/README.md
CREATE libs/my-slos/.babelrc
CREATE libs/my-slos/package.json
CREATE libs/my-slos/src/index.ts
CREATE libs/my-slos/src/lib/my-slos.spec.ts
CREATE libs/my-slos/src/lib/my-slos.ts
CREATE libs/my-slos/tsconfig.json
CREATE libs/my-slos/tsconfig.lib.json
UPDATE tsconfig.base.json
UPDATE workspace.json
UPDATE nx.json
UPDATE package.json
CREATE .eslintrc.json
UPDATE .vscode/extensions.json
CREATE libs/my-slos/.eslintrc.json
CREATE jest.config.js
CREATE jest.preset.js
CREATE libs/my-slos/jest.config.js
CREATE libs/my-slos/tsconfig.spec.json
CREATE libs/my-slos/src/lib/slo-mappings/cost-efficiency.slo-mapping.ts
CREATE libs/my-slos/src/lib/init-polaris-lib.ts
npm WARN deprecated request-promise-native@1.0.9: request-promise-native has been deprecated because it extends the now deprecated request package, see https://github.com/request/request/issues/3142
npm WARN deprecated request@2.88.2: request has been deprecated, see https://github.com/request/request/issues/3142
npm WARN deprecated har-validator@5.1.5: this library is no longer supported
[ ] .....] / extract:abab: 511 extract word-wrap@1.2.3
```

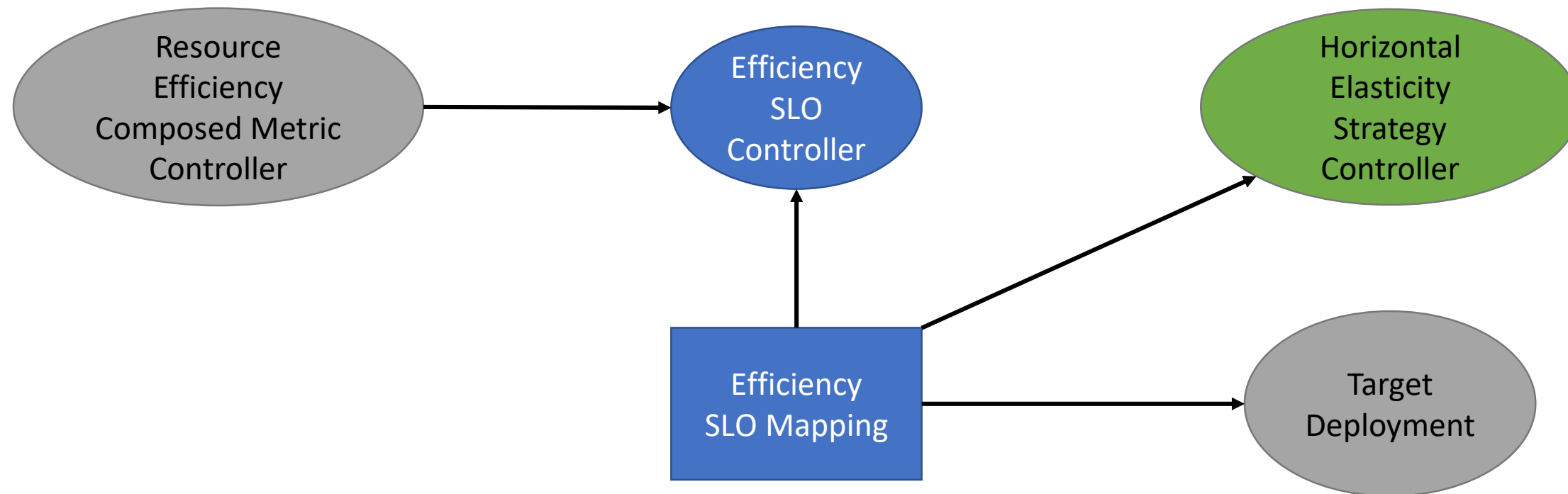
Polaris Demo

Horizontal scaling of a Deployment, based on Resource Efficiency SLO computed from metrics from the Google Cluster Data 2011 [2]



[2] <https://research.google/tools/datasets/cluster-workload-traces/>

Polaris Demo Components





RAINBOW

Questions?



RAINBOW

Demo

Thank you!



👉 <https://rainbow-h2020.eu/>  @RainbowH2020

Polaris Framework: <https://polaris-slo-cloud.github.io>