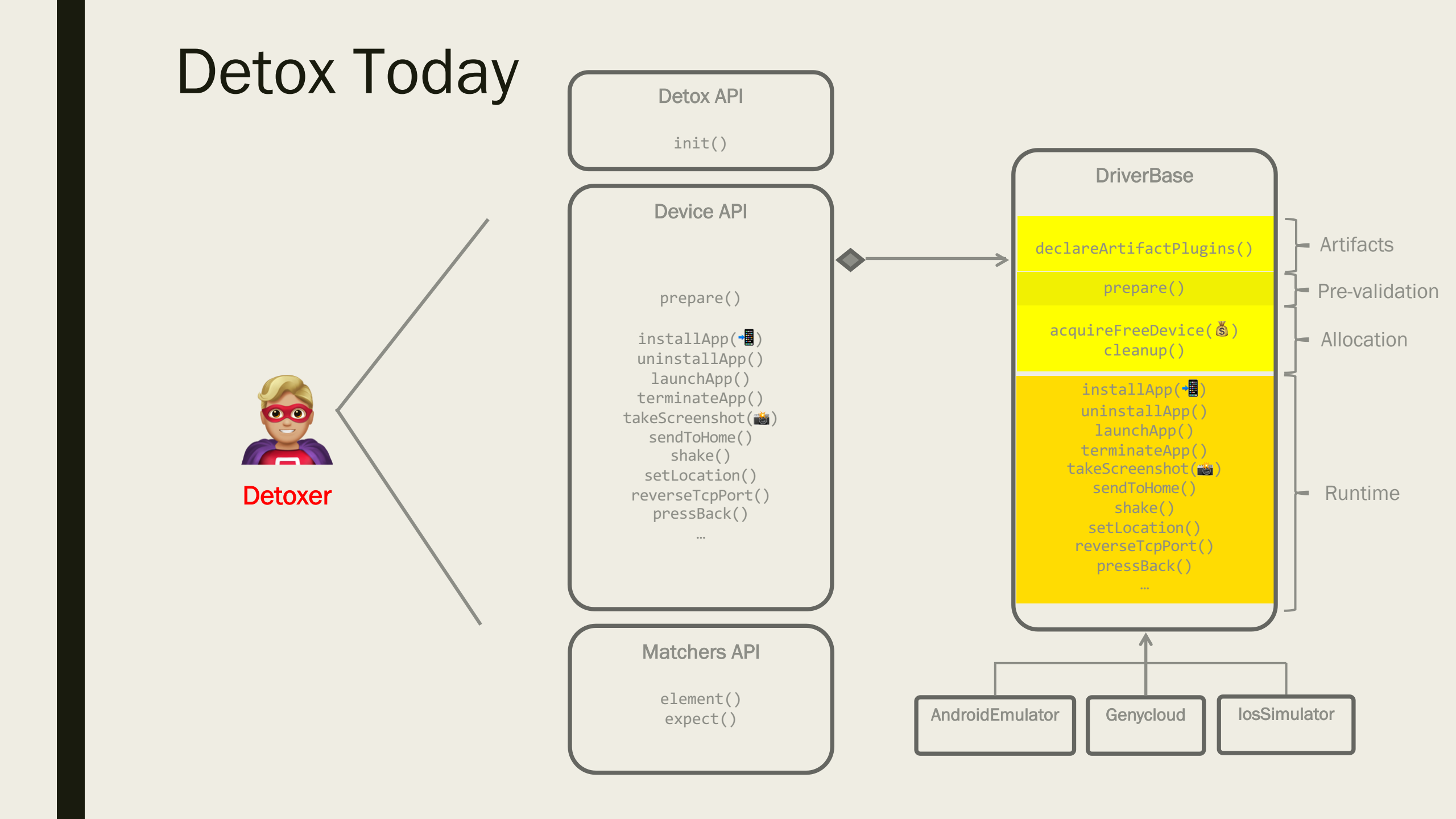
PRE-MULTI-APPS REFACTOR

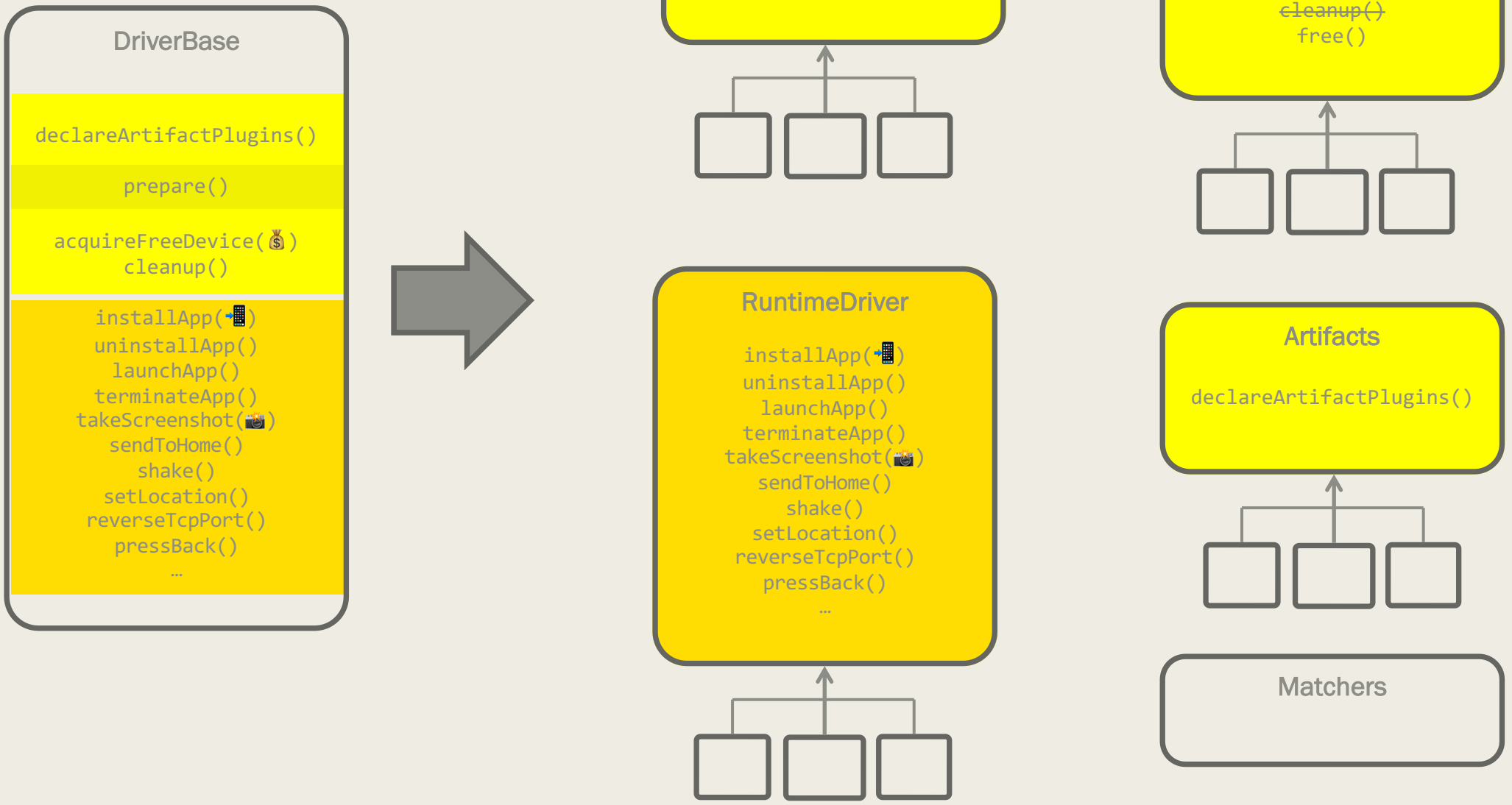
Request For Comment

Scope 1: Driver Decompose

Detox Today


Detox API
init()
Device API
prepare()
installApp(📲)
uninstallApp()
launchApp()
terminateApp()
takeScreenshot(📸)
sendToHome()
shake()
setLocation()
reverseTcpPort()
pressBack()
…
Matchers API
element()
expect()
Detoxer
DriverBase
declareArtifactPlugins()
prepare()
acquireFreeDevice(💰)
cleanup()
installApp(📲)
uninstallApp()
launchApp()
terminateApp()
takeScreenshot(📸)
sendToHome()
shake()
setLocation()
reverseTcpPort()
pressBack()
…
Artifacts
Pre-validation
Allocation
Runtime
AndroidEmulator
Genycloud
IosSimulator

Detox Refactored



Why?

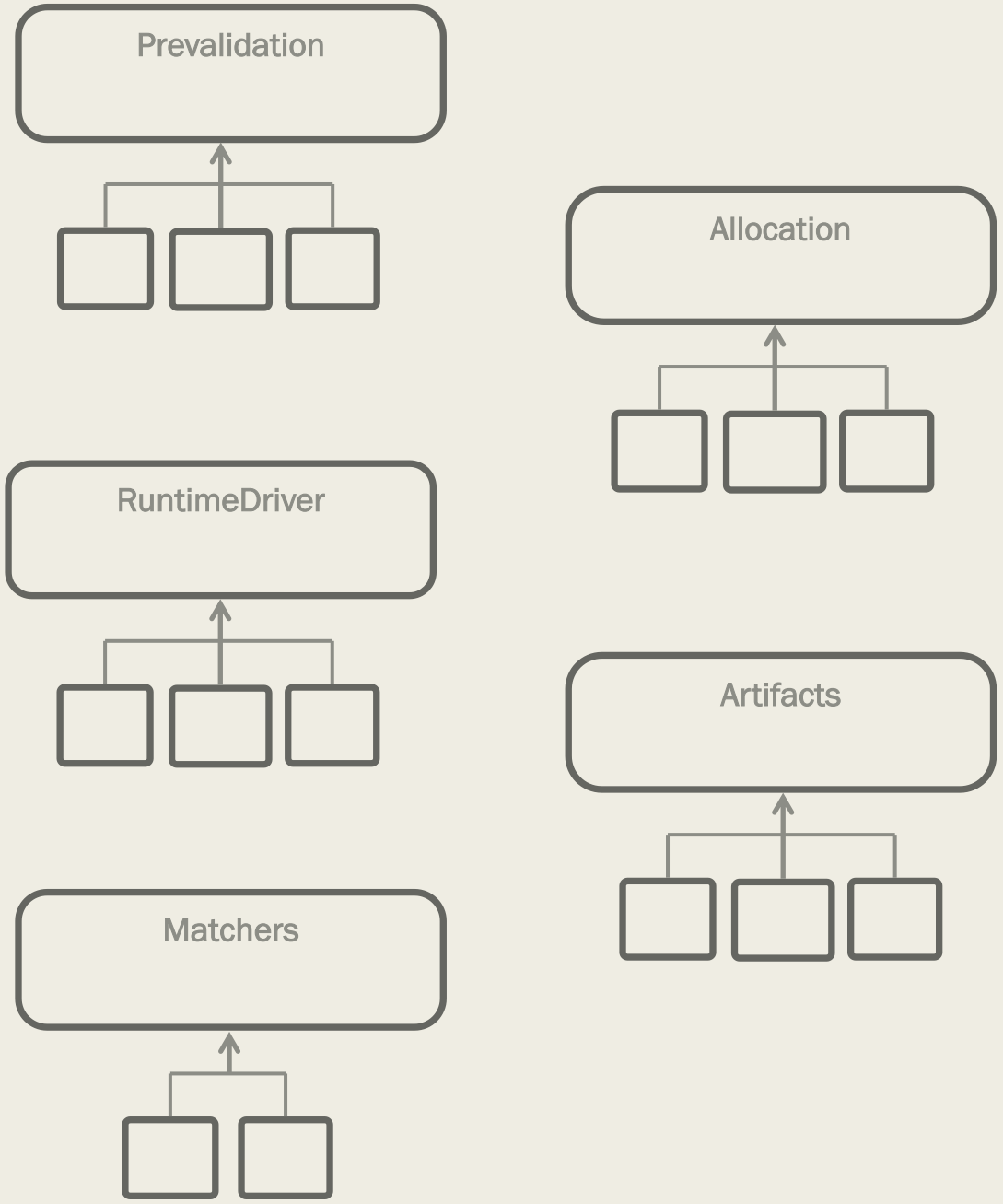
- Maintainability
  - *Well separated Concerns*
  - *Greatly simplified unit tests + better coverage*
- Flexibility
  - *DAS*
- Clear, “compile”-time safe `deviceId` (i.e. lifecycle) management
  - *`deviceAllocation.allocate()` ⇨ `deviceCookie` (e.g. `AndroidEmulatorCookie`)*
  - *`cookie` ⇨ `new EmulatorRuntimeDevice(cookie.adbName, cookie.avdName)`*
- ~~Multiple apps~~
  - *(easier to move forward with fundamental changes)*

Why “not”?

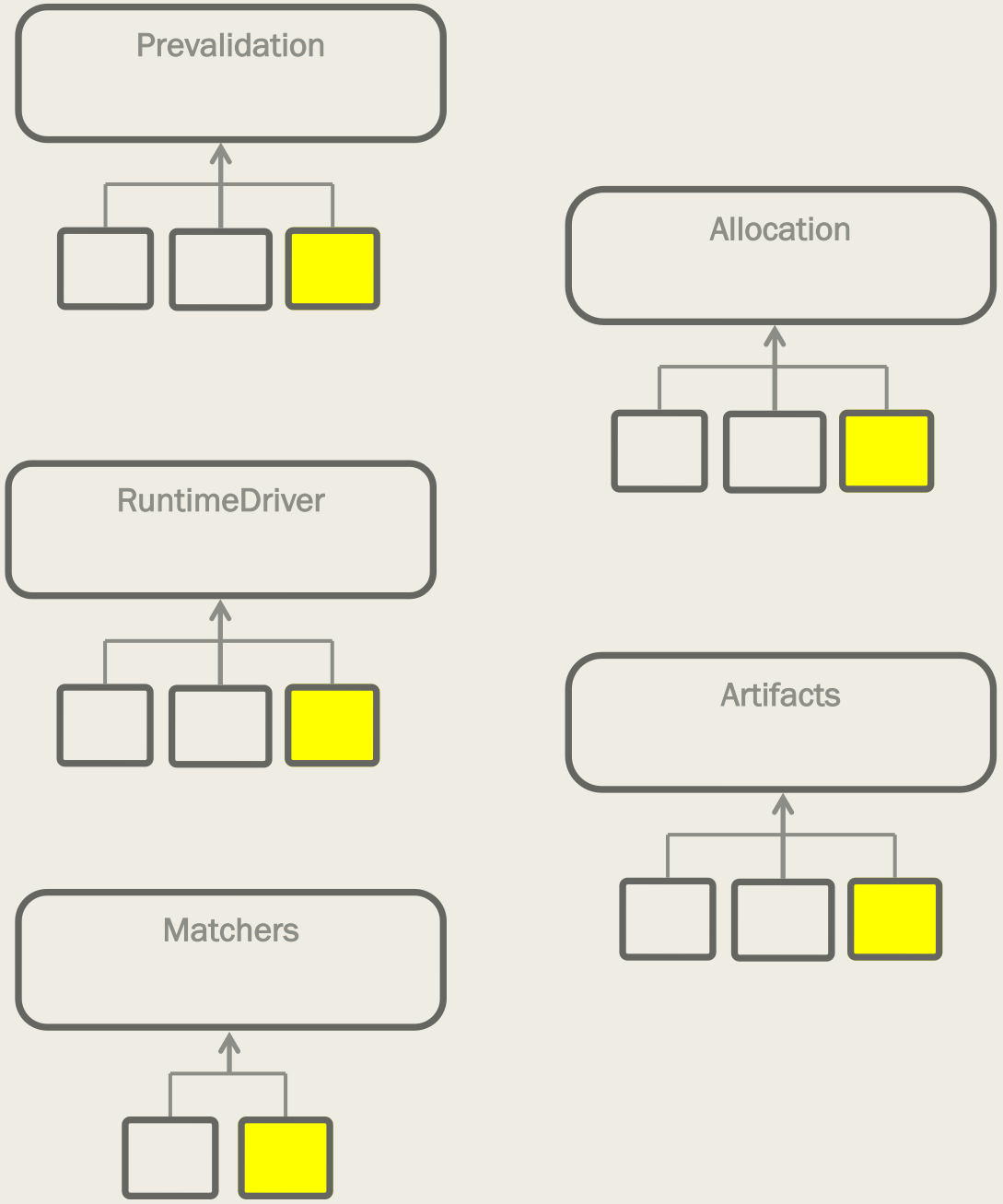
- Factories
- Environment factory
  - *A factory of factories*

```
const {
  envValidatorFactory,
  deviceAllocatorFactory,
  artifactsManagerFactory,
  matchersFactory,
  runtimeDeviceFactory,
} = environmentFactory.createFactories(this._deviceConfig);
```

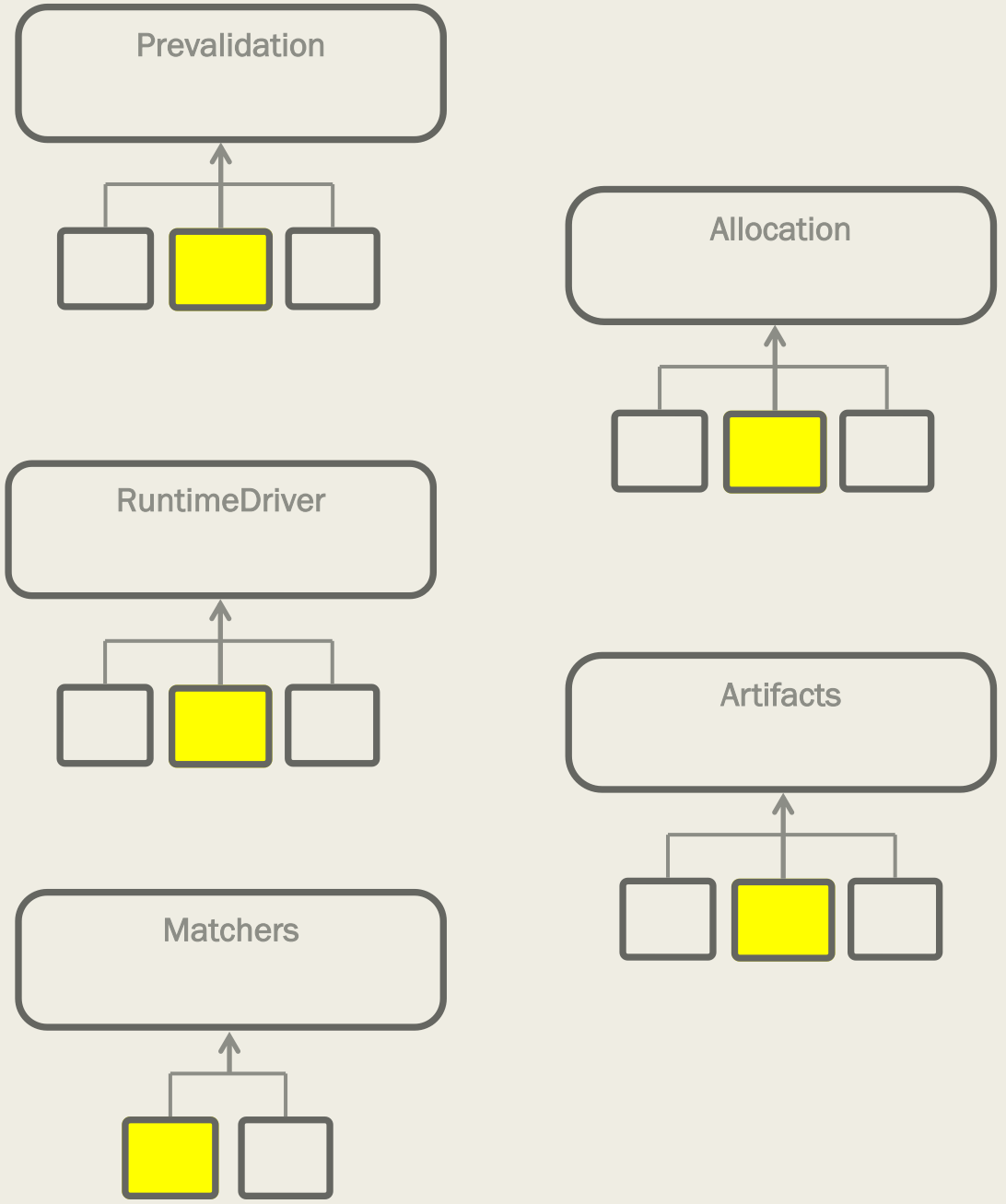
```
const {
  envValidatorFactory,
  deviceAllocatorFactory,
  artifactsManagerFactory,
  matchersFactory,
  runtimeDeviceFactory,
} = environmentFactory.createFactories(this._deviceConfig);
```



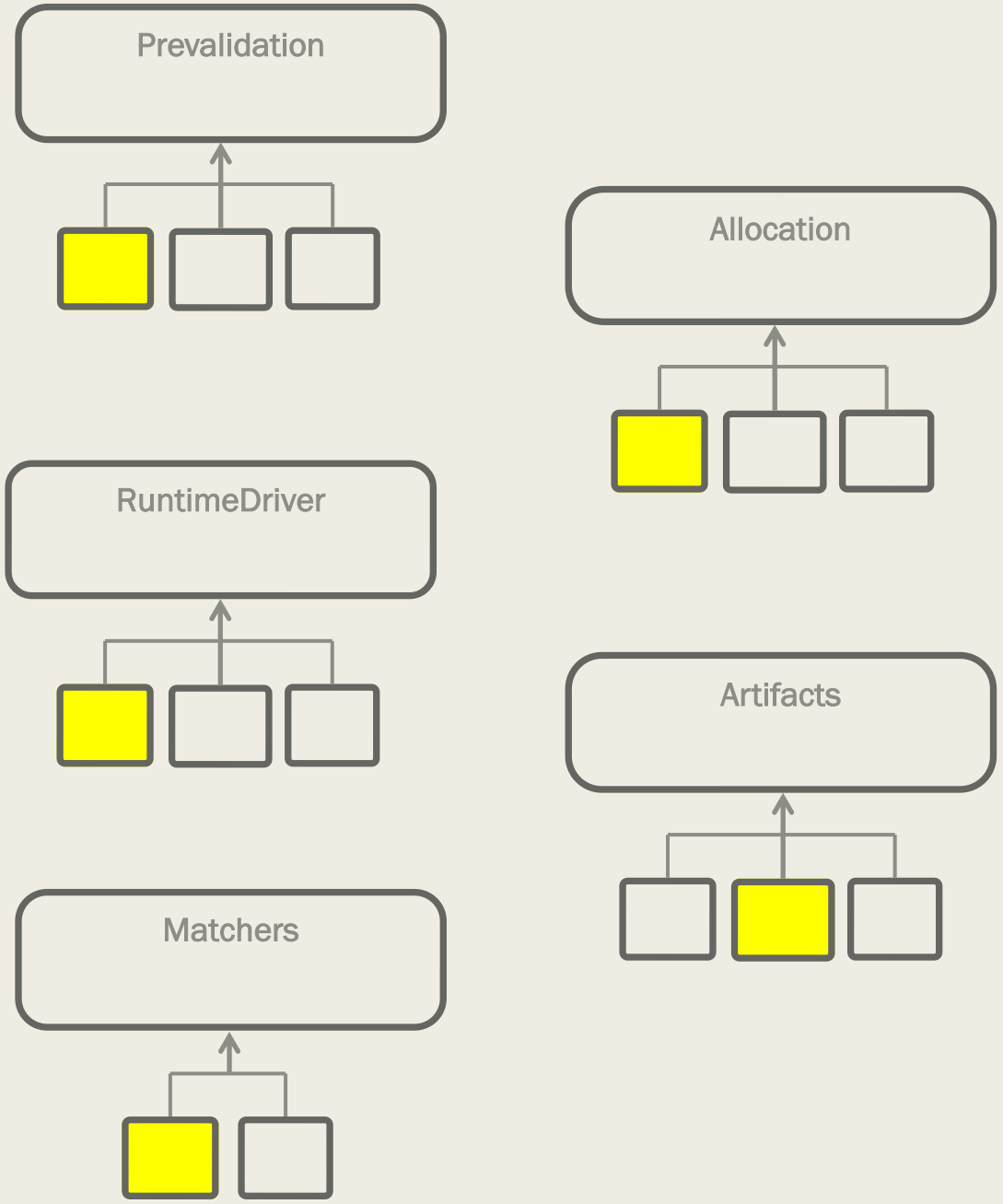
```
deviceConfig.type ===
    “ios.simulator”
```



```
deviceConfig.type ===
    “android.emulator”
```



```
deviceConfig.type ===
    “android.genycloud”
```



Scope 2: Driver State

(Runtime) Driver State

- **Fundamentals**
  - `device.deviceId` is no more
  - Device is stateless; Each runtime-driver holds necessary info, explicitly:
    - Android: ADB name
    - Android Emulator: ADB name, AVD name
    - Genycloud: Instance DTO
    - iOS Simulator: Simulator UDID
  - `device.name` is inferred by the concrete driver (e.g. `` `${adbName} (${avdName})` ``)
- **Why?**
  - Simplified API: no more “multi-typed” `deviceId` parameter in each call
  - (Multi-apps) The currently selected app becomes part of the managed state

Scope 3: IoC

Detox Today

```
class GenyCloudDriver extends AndroidDriver {
  constructor(config) {
    super(config);
    this._name = 'Unspecified GMSaaS Emulator';

    this._exec = new GenyCloudExec(environment.getGmsaasPath());
    const instanceNaming = new InstanceNaming();
    const deviceRegistry = GenyDeviceRegistryFactory.forRuntime();
    const deviceCleanupRegistry = GenyDeviceRegistryFactory.forGlobalShutdown();

    const recipeService = new RecipesService(this._exec, logger);
    const instanceLookupService = new InstanceLookupService(this._exec, instanceNaming, deviceRegistry);
    const instanceLifecycleService = new InstanceLifecycleService(this._exec, instanceNaming);
    const instanceLauncher = new InstanceLauncher(this._instanceLifecycleService, deviceCleanupRegistry, this.emitter);
    this._recipeQuerying = new RecipeQuerying(recipeService);
    this._instanceAllocation = new GenyCloudInstanceAllocation({ deviceRegistry, instanceLookupService, instanceLifecycleService, instanceLauncher, eventEmitter: this.emitter });
    this._instanceLifecycleService = instanceLifecycleService;
    this._instanceLauncher = instanceLauncher;

    this._authService = new AuthService(this._exec);
  }
```

Detox Refactored

- Inverted control:
  1. *To the associated factory*
  2. *To a service-locator (~intermediate solution)*

```
class AndroidRuntimeDriverFactory extends RuntimeDriverFactory {
  _createDependencies(commonDeps) {
    const serviceLocator = require('../../../servicelocator/android');
    const adb = serviceLocator.adb();
    const aapt = serviceLocator.aapt();
    const fileXfer = serviceLocator.fileXfer();
    const devicePathBuilder = serviceLocator.devicePathBuilder();

    const AppInstallHelper = require('../../common/drivers/android/tools/AppInstallHelper');
    const AppUninstallHelper = require('../../common/drivers/android/tools/AppUninstallHelper');
    const MonitoredInstrumentation = require('../../common/drivers/android/tools/MonitoredInstrumentation');

    return {
      ...commonDeps,
      adb,
      aapt,
      fileXfer,
      devicePathBuilder,
      appInstallHelper: new AppInstallHelper(adb, fileXfer),
      appUninstallHelper: new AppUninstallHelper(adb),
      instrumentation: new MonitoredInstrumentation(adb),
    }
  }
}
```

```
class GenycloudRuntimeDriverFactory extends AndroidRuntimeDriverFactory {
  _createDriver(deviceCookie, deps) {
    const { instance } = deviceCookie;
    return new GenycloudRuntimeDriver(instance, deps);
  }
}
```

```
class GenyCloudDriver extends AndroidDriver {
  /**
   * @param instance { GenyInstance } The DTO associated with the cloud instance
   * @param deps { Object }
   */
  constructor(instance, deps) {
    super(instance.adbName, deps);
    this.instance = instance;
  }
```

Caveats

Caveats

- External drivers is a breaking change
  - *Major version?*

Future Plans

Future plans

- DAS
- External Device separation
  - *Eliminate access to private methods*
- Multiple apps