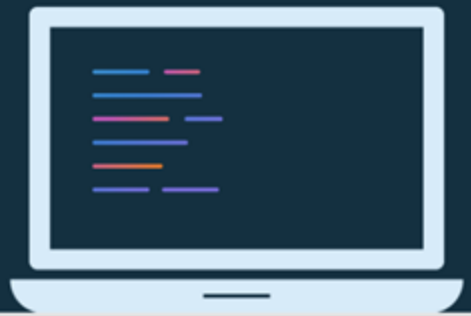
Lesson 12: Repository pattern and WorkManager



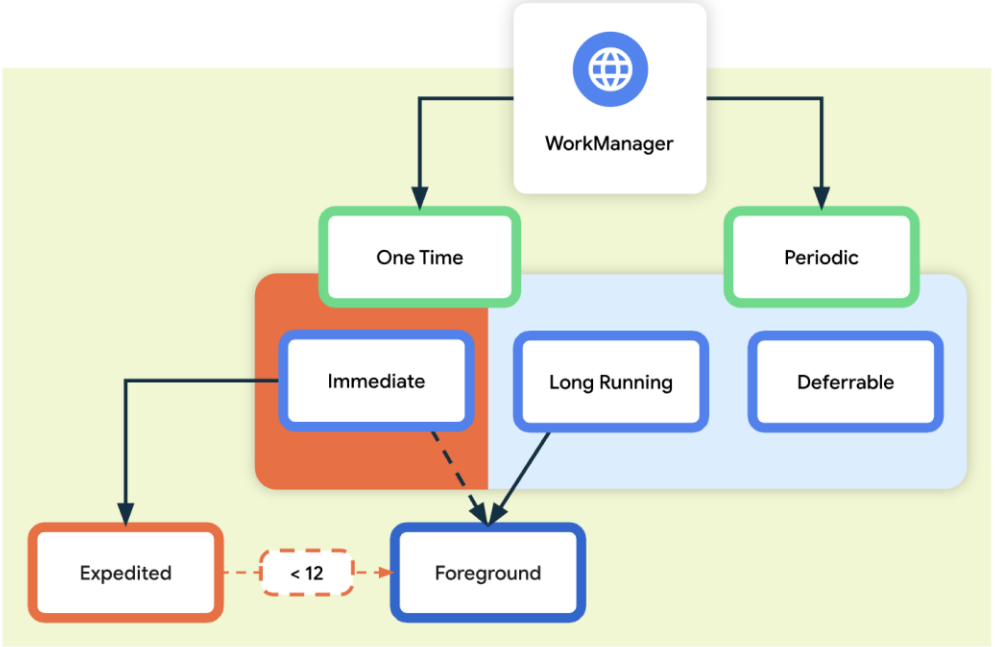
WorkManager

WorkManager

- Android Jetpack architecture component
- Recommended solution to execute background work (immediate or deferred)
- Opportunistic and guaranteed execution
- Execution can be based on certain conditions

WorkManager

- It's for persistent work
- For non-persistent work use coroutines



Declare WorkManager dependencies

```
implementation "androidx.work:work-runtime-ktx:$work_version"
```

Important classes to know

- `Worker` - does the work on a background thread, override `doWork()` method
- `WorkRequest` - request to do some work
- `Constraint` - conditions on when the work can run
- `WorkManager` - schedules the `WorkRequest` to be run

Define the work

```
class UploadWorker(appContext: Context, workerParams: WorkerParameters) :
        Worker(appContext, workerParams) {

    override fun doWork(): Result {

        // Do the work here. In this case, upload the images.
        uploadImages()

        // Indicate whether work finished successfully with the Result
        return Result.success()
    }
}
```

Extend CoroutineWorker instead of Worker

```
class UploadWorker(appContext: Context, workerParams: WorkerParameters) :
        CoroutineWorker(appContext, workerParams) {

    override suspend fun doWork(): Result {

        // Do the work here (in this case, upload the images)
        uploadImages()

        // Indicate whether work finished successfully with the Result
        return Result.success()
    }
}
```

WorkRequests

- Can be scheduled to run once or repeatedly
  - `OneTimeWorkRequest`
  - `PeriodicWorkRequest`
- Persisted across device reboots
- Can be chained to run sequentially or in parallel
- Can have constraints under which they will run

Schedule a OneTimeWorkRequest

Create `WorkRequest`:

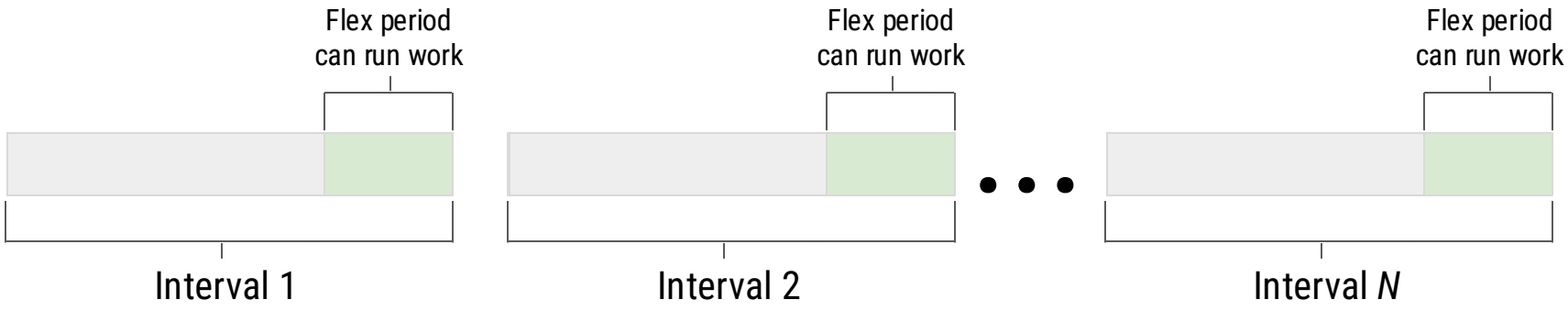
```
val uploadWorkRequest: WorkRequest =
   OneTimeWorkRequestBuilder<UploadWorker>()
       .build()
```

Add the work to the `WorkManager` queue:

```
WorkManager.getInstance(myContext)
    .enqueue(uploadWorkRequest)
```

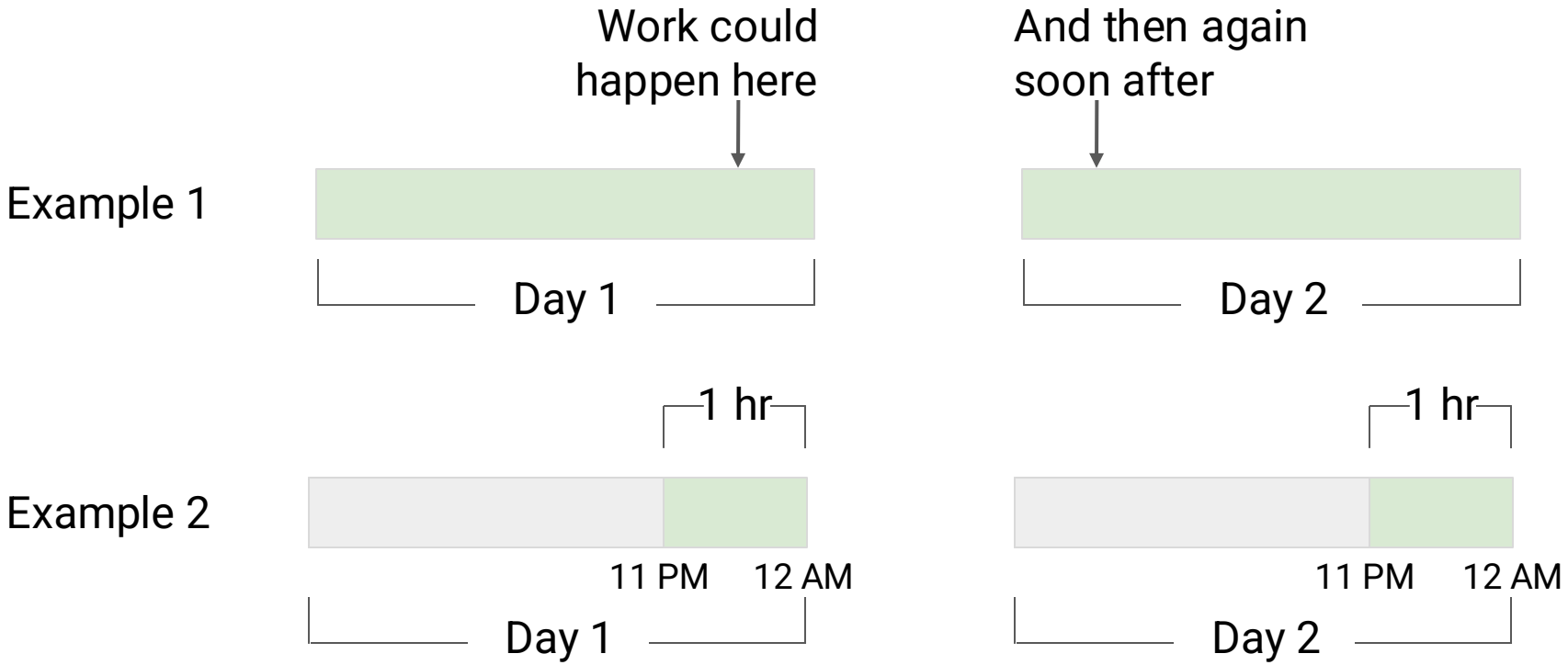
Schedule a PeriodicWorkRequest

- Set a repeat interval
- Set a flex interval (optional)



Specify an interval using `TimeUnit` (e.g., `TimeUnit.HOURS`, `TimeUnit.DAYS`)

Flex interval



PeriodicWorkRequest example

```
val repeatingRequest = PeriodicWorkRequestBuilder<RefreshDataWorker>(
        1, TimeUnit.HOURS,     // repeatInterval
        15, TimeUnit.MINUTES   // flexInterval
    ).build()
```

Enqueue periodic work

```
WorkManager.getInstance().enqueueUniquePeriodicWork(
    "Unique Name",
    ExistingPeriodicWorkPolicy.KEEP, // or REPLACE
    repeatingRequest
)
```

Work input and output

Define Worker with input and output

```
class MathWorker(context: Context, params: WorkerParameters):
      CoroutineWorker(context, params)  {

    override suspend fun doWork(): Result {
        val x = inputData.getInt(KEY_X_ARG, 0)
        val y = inputData.getInt(KEY_Y_ARG, 0)
        val result = computeMathFunction(x, y)
        val output: Data = workDataOf(KEY_RESULT to result)
        return Result.success(output)
    }
}
```

Result output from doWork()

Result status	Result status with output
`Result.success()`	`Result.success(output)`
`Result.failure()`	`Result.failure(output)`
`Result.retry()`	

Send input data to Worker

```
val complexMathWork = OneTimeWorkRequest<MathWorker>()
    .setInputData(
        workDataOf(
            "X_ARG" to 42,
            "Y_ARG" to 421,
        )
    ).build()

WorkManager.getInstance(myContext).enqueue(complexMathWork)
```

WorkRequest constraints

Constraints

- `setRequiredNetworkType`
- `setRequiresBatteryNotLow`
- `setRequiresCharging`
- `setRequiresDeviceIdle`

Constraints example

```
val constraints = Constraints.Builder()
    .setRequiredNetworkType(NetworkType.UNMETERED)
    .setRequiresCharging(true)
    .setRequiresBatteryNotLow(true)
    .setRequiresDeviceIdle(true)
    .build()

val myWorkRequest: WorkRequest = OneTimeWorkRequestBuilder<MyWork>()
    .setConstraints(constraints)
    .build()
```