映像制作パイプラインにおける**PDG**の役割り

Pavel Smirnov
株式会社グリオグルーヴ・LiNDAチーム

PDGとは

凄そうだけど、なんか今ー、、？

使い道

- プロシージャルコンテント作り
- レンダーファームの扱いをより効率化
- ウェッジング
- パイプライン
- Houdiniコミュニティーがこれから発見する使い方

使い道

- プロシージャルコンテント作り
- レンダーファームの扱いをより効率化
- ウェッジング
- **パイプライン　←**
- Houdiniコミュニティーがこれから発見する使い方

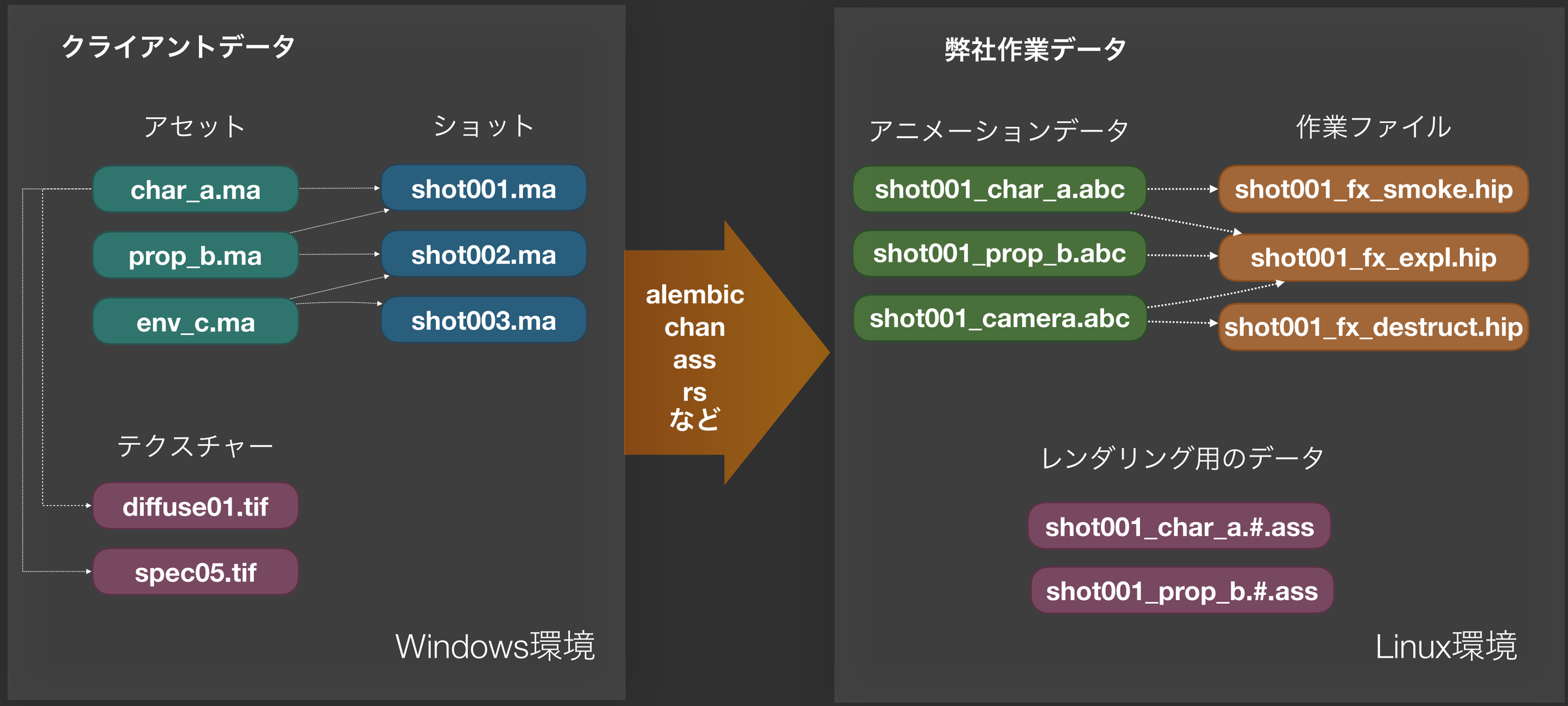
注意

以前Pythonが書けないと出来なかったことが簡単になってきた

- 一瞬でファームに500本のシミュレーションジョブを投げる
- ファイルを分散コピーしすぎで、サーバーを落とす
- 短期でデータをウン十テラ作っちゃう
- Mayaファイルを探して全て消す

本気で実行する前、
必ずテストデータに使ってみよう

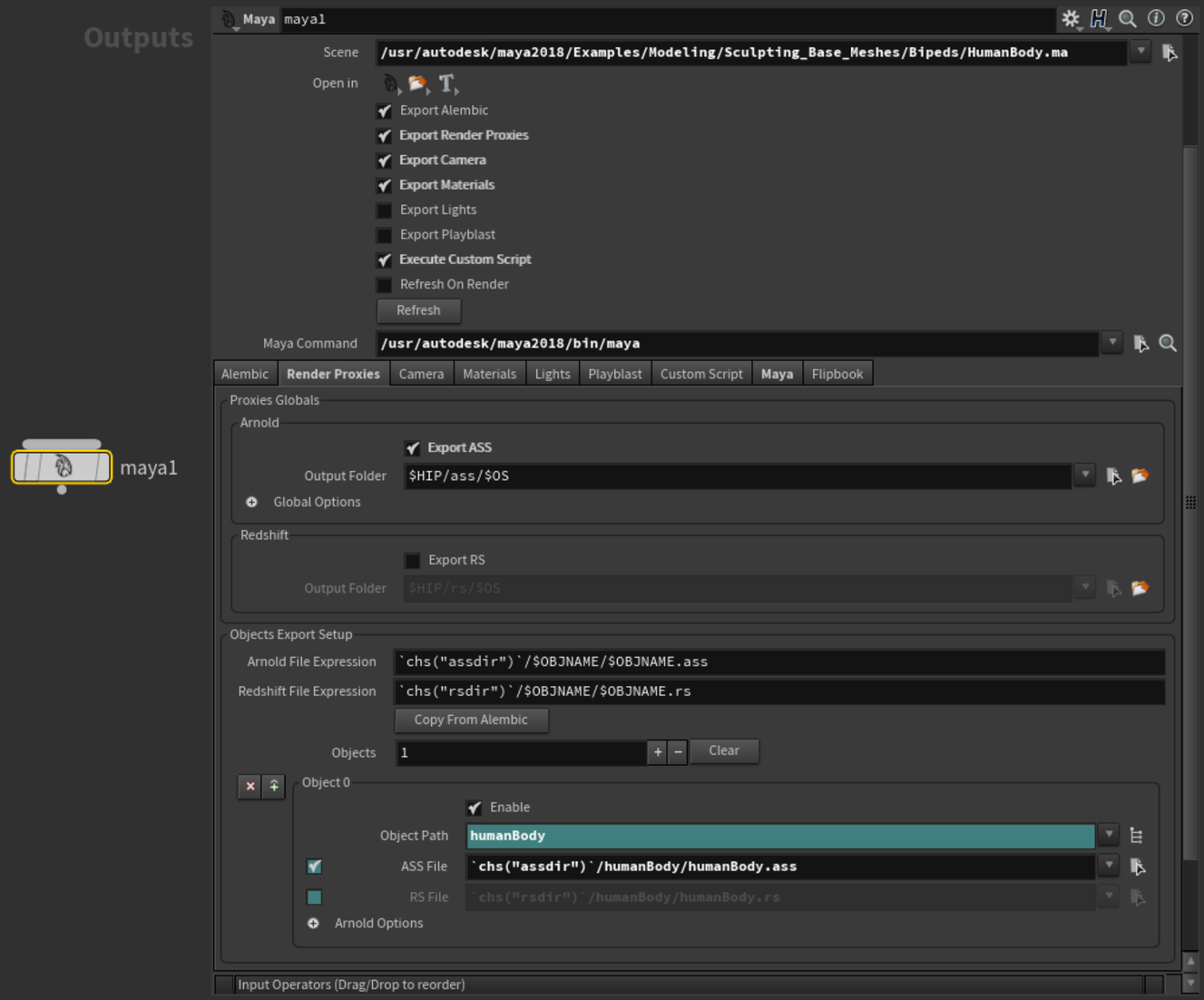
よくあるパイプラインタスク：データ変換と読み込み



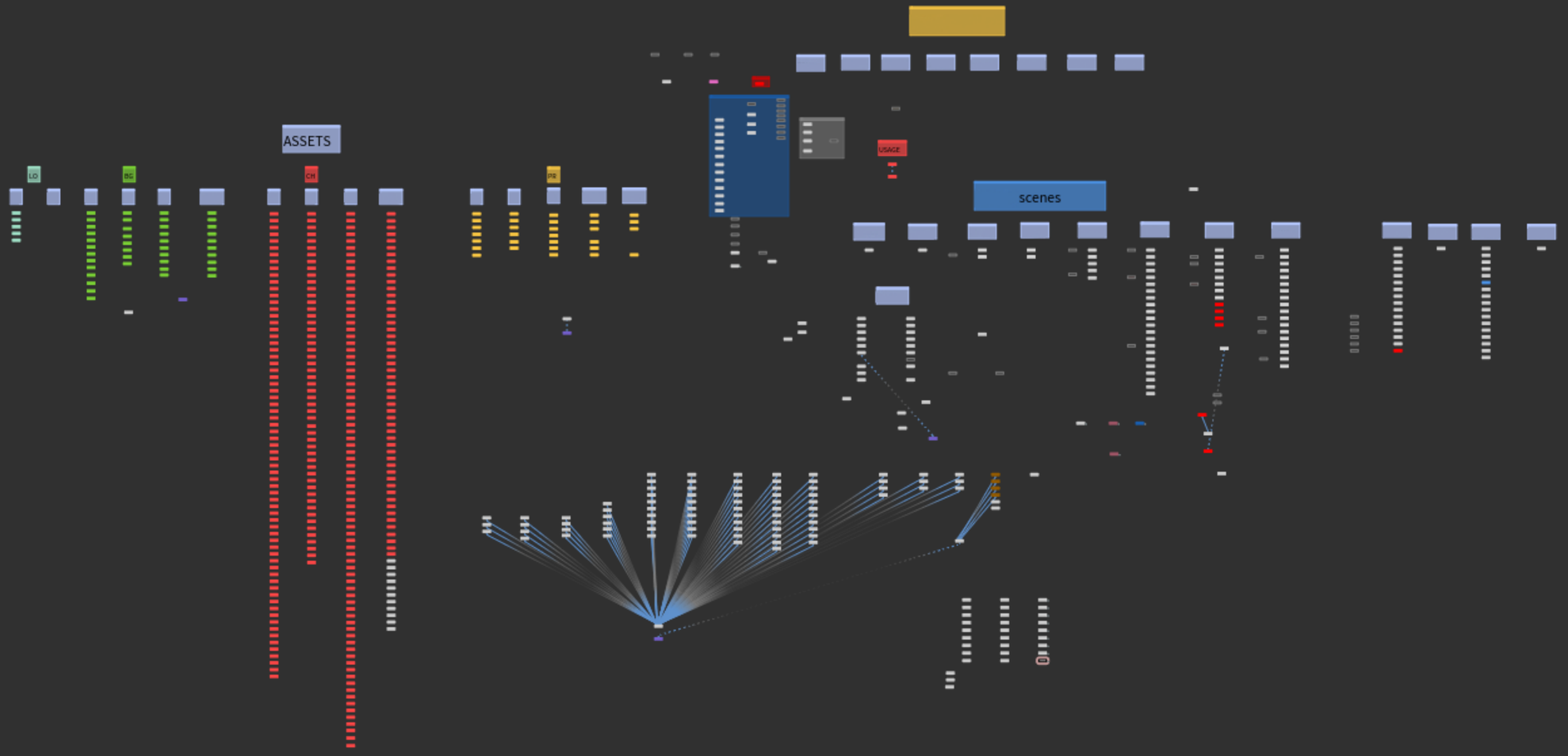
問題

- WinとLinuxのパス
- リファレンス再読み込み
- テクスチャーパス
- 個別に書き出すものが多い
- データが足りなかった場合の対応
- ショットの数が多ければ、手作業では無理

ROPベースパイプライン



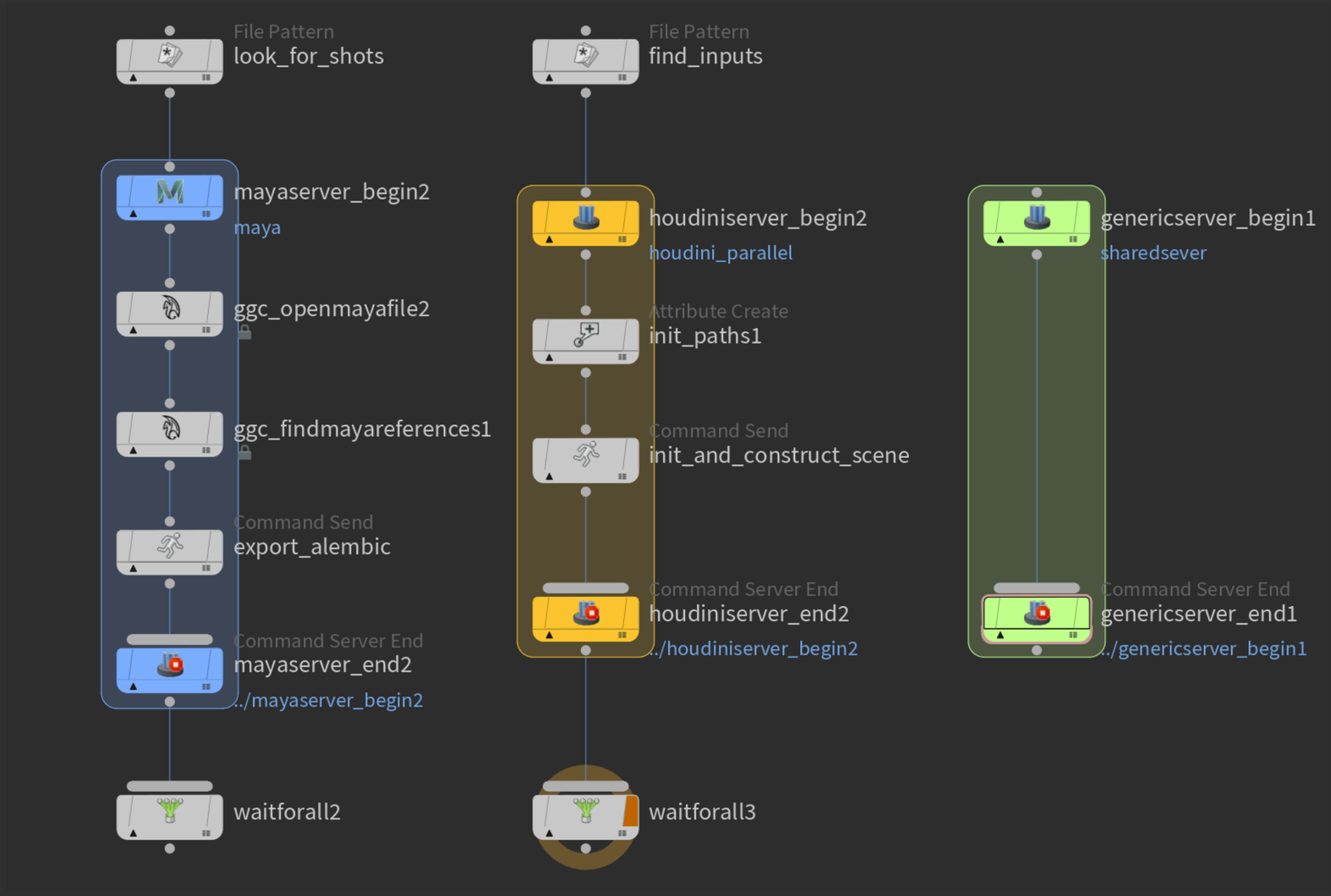
あるプロジェクトの管理シーン



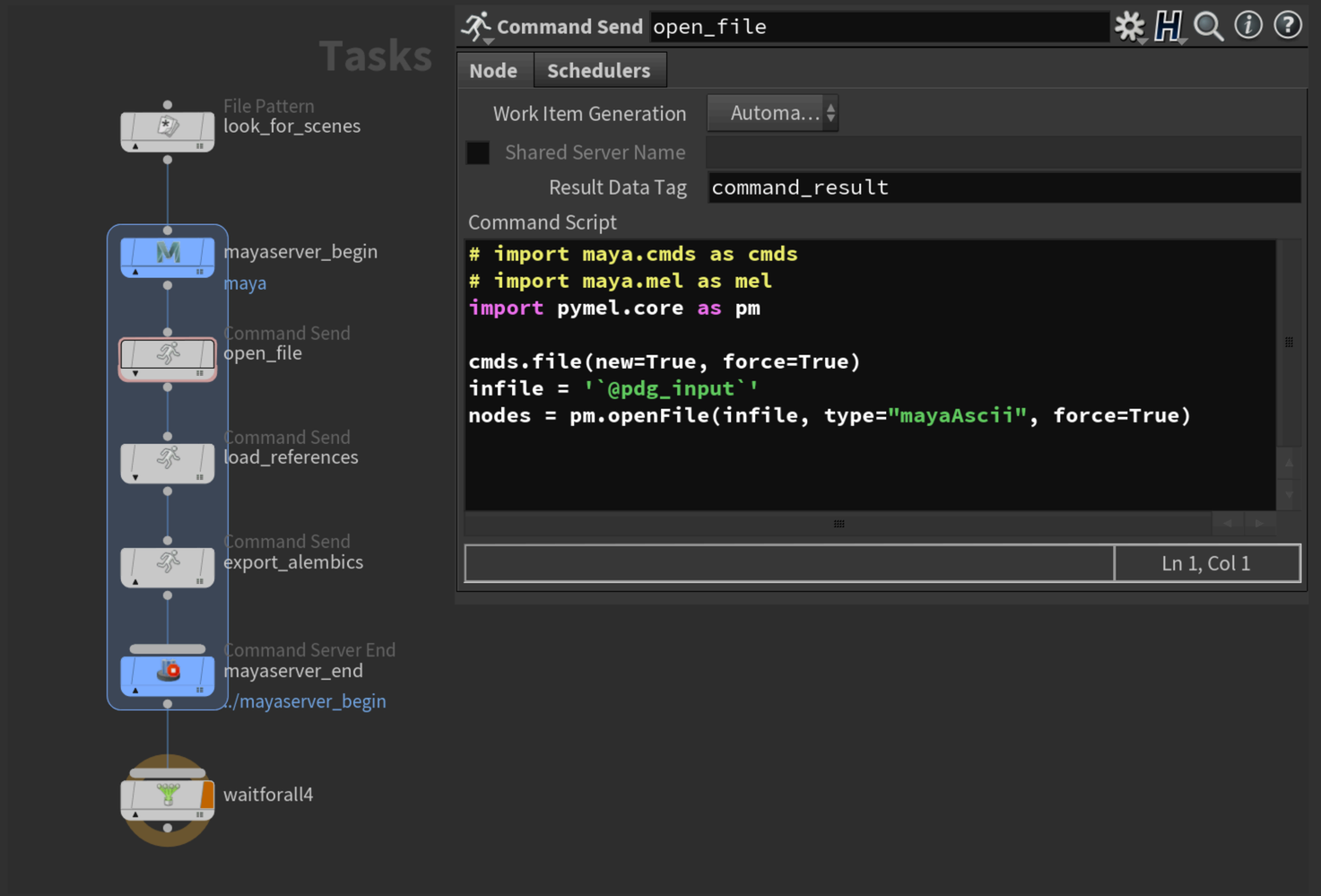
ROPベースパイプラインのデメリット

- Pythonヘビー
- カスタムROPのデバッグやアップデートに会社のリソースをかけなければならない
- 一つのシーンの扱いはプロシージャルではない
- フレキシビリティーが足りない
- 大きめのプロジェクトの管理シーンは恐ろしくなる傾向がある
- 最低Houdini Coreのライセンスが必要

Command Chains



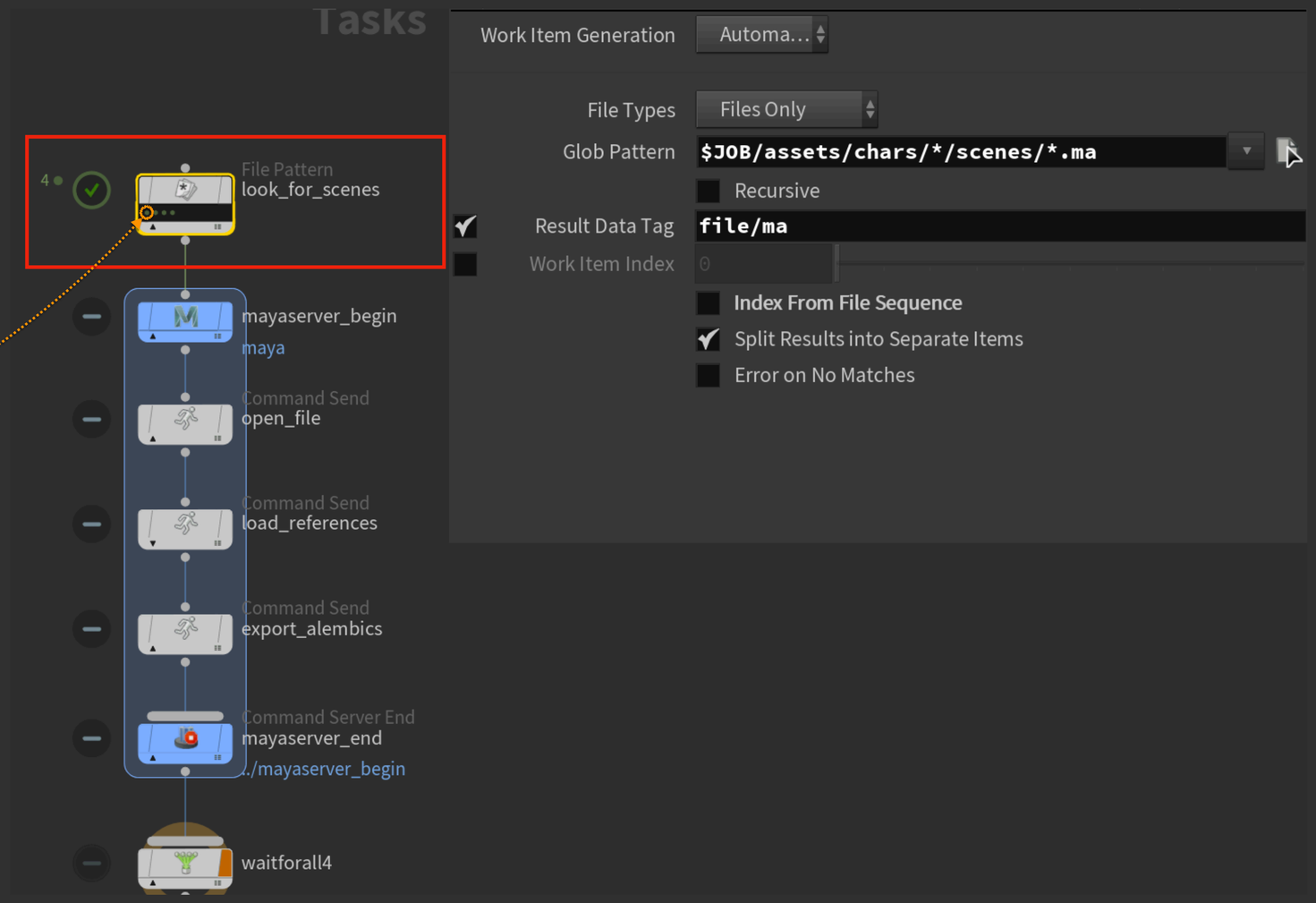
Maya Command Chain



Maya Command Chain

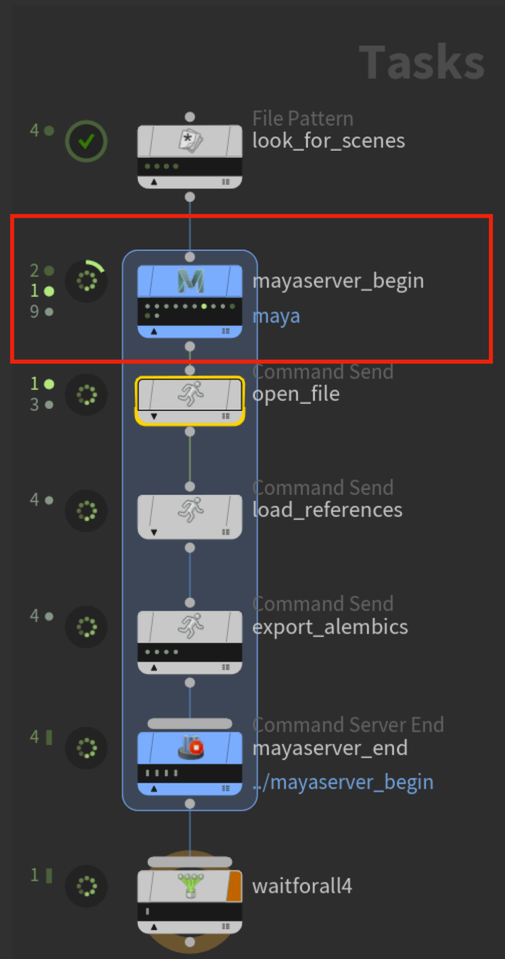
ディスクでMayaシーンを探して
ルートの**ワークアイテム**を作る

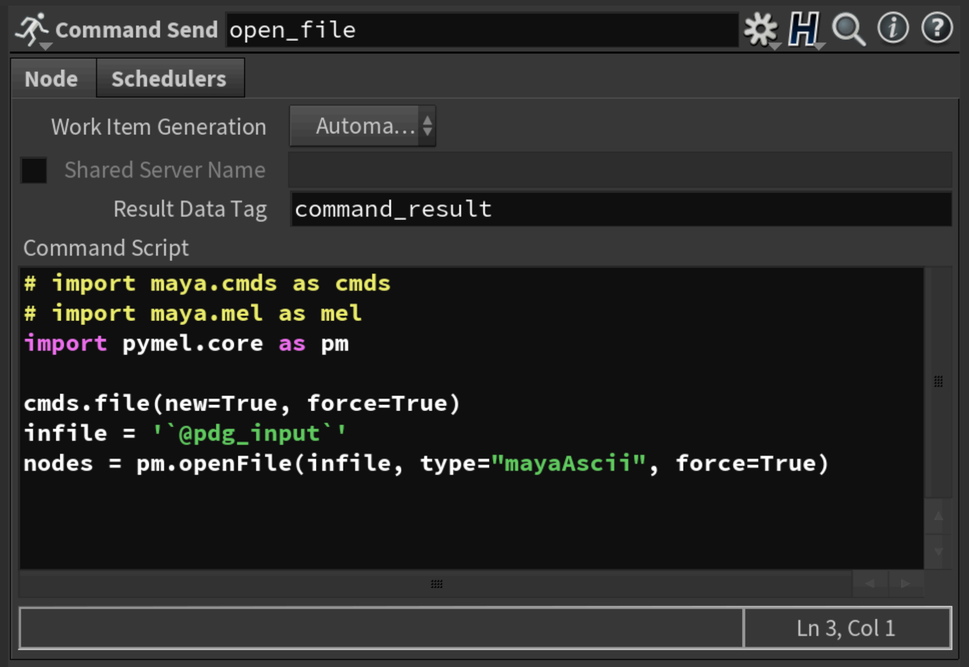
これがワークアイテム



Maya Command Chain

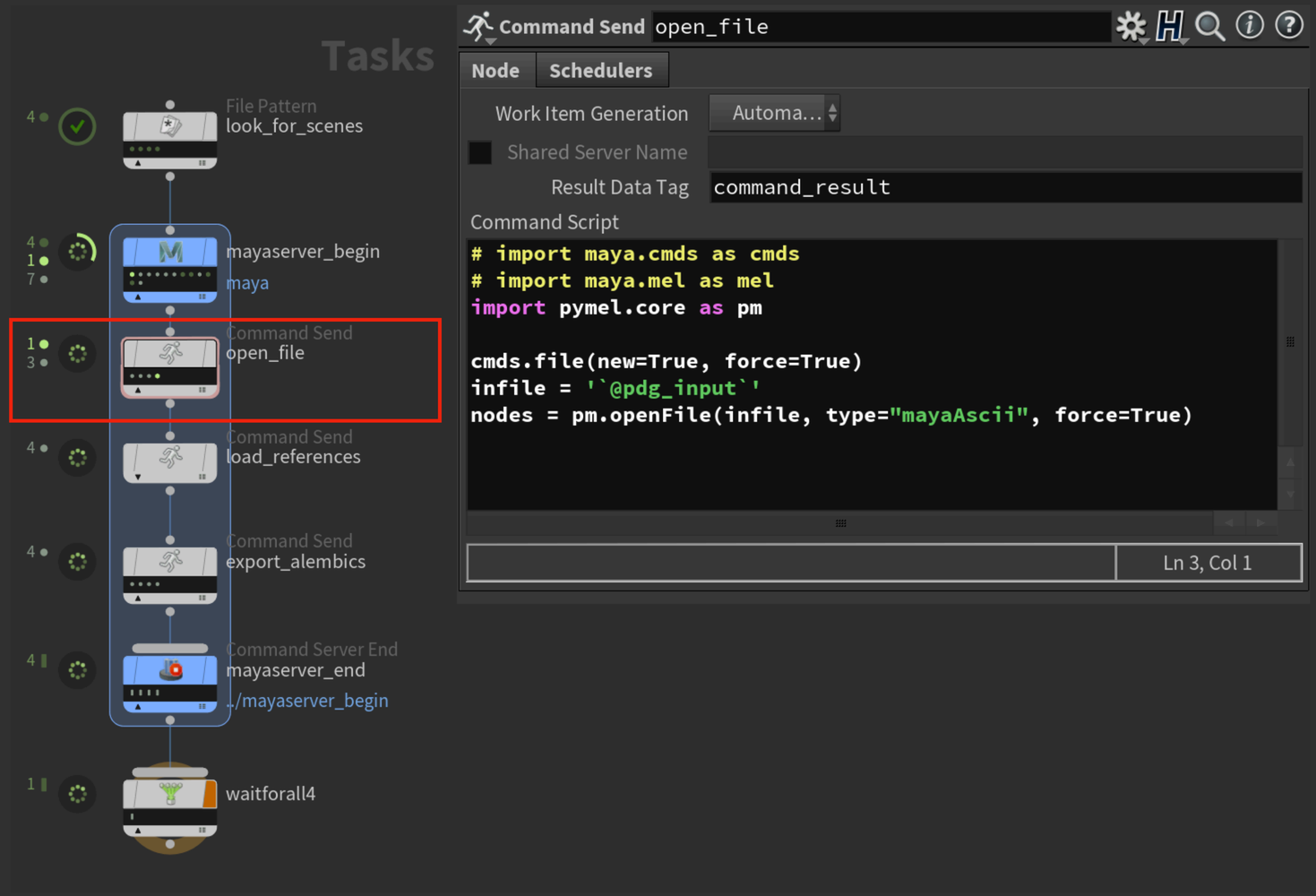
裏で**Maya**を立ち上げて





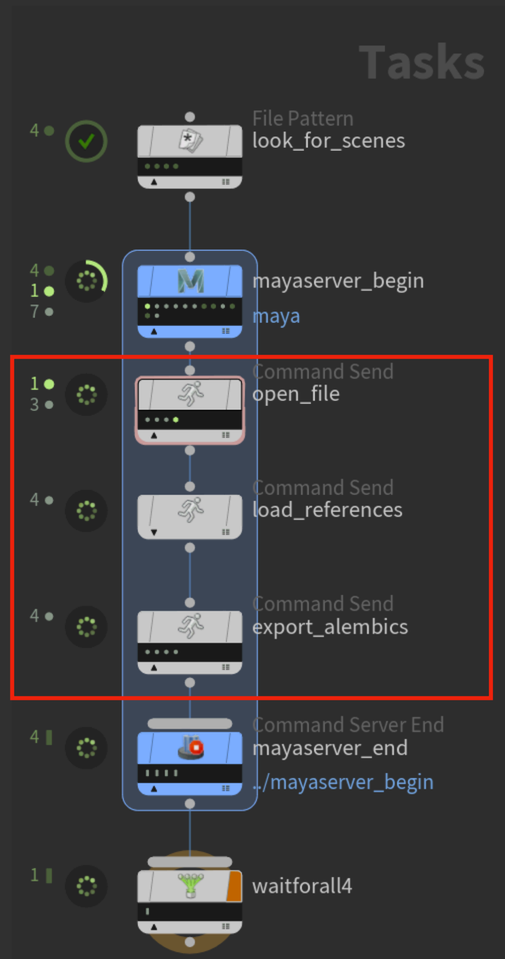
Maya Command Chain

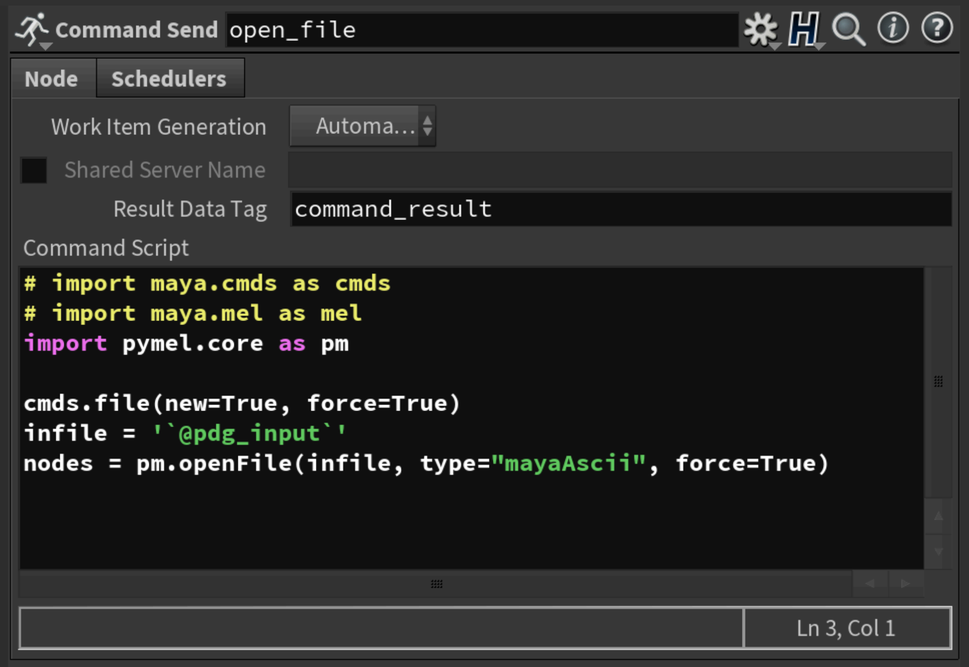
各**アイテム**に対して
MayaにMELかPythonの
スクリプトを送って実行する



Maya Command Chain

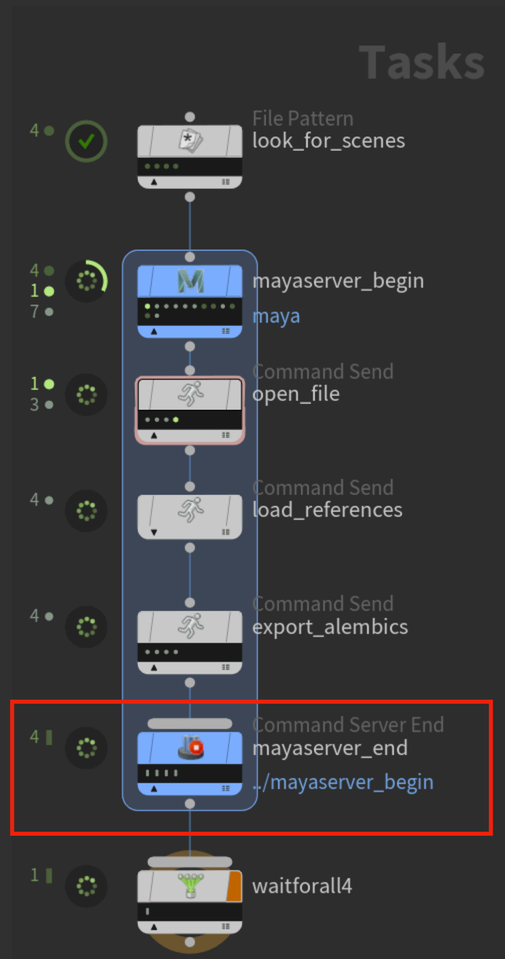
枠の中のノードはMayaの中で行なっている

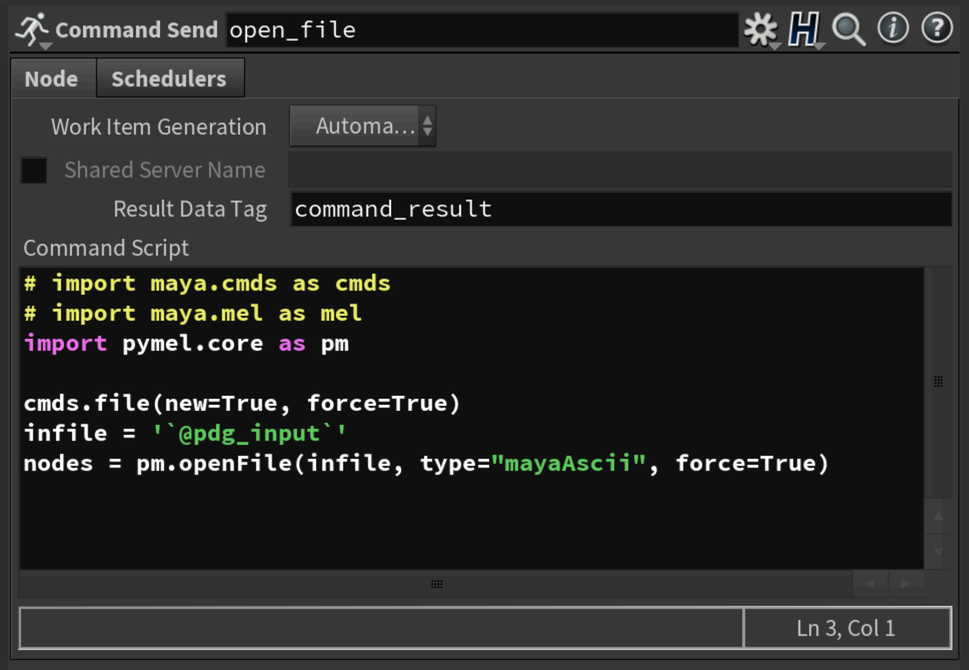




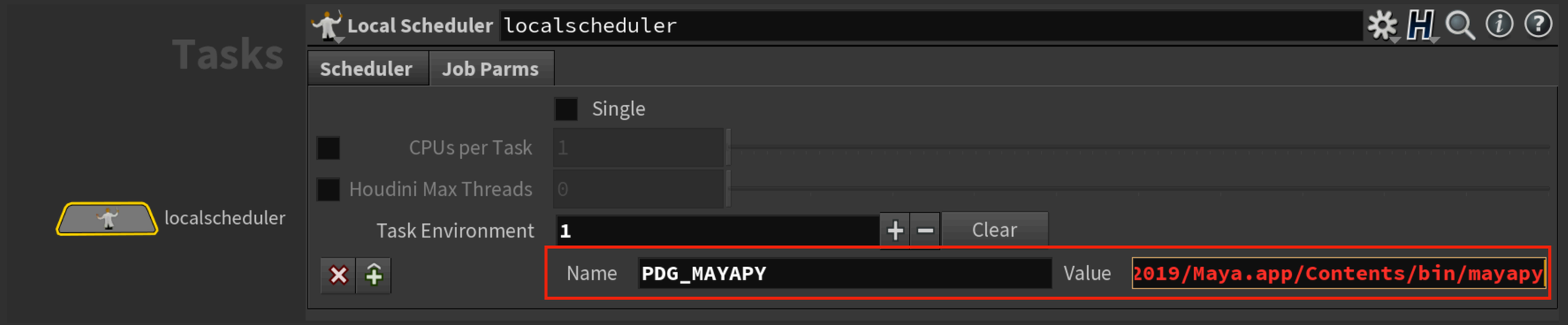
Maya Command Chain

結果をまとめて
Mayaを落とす



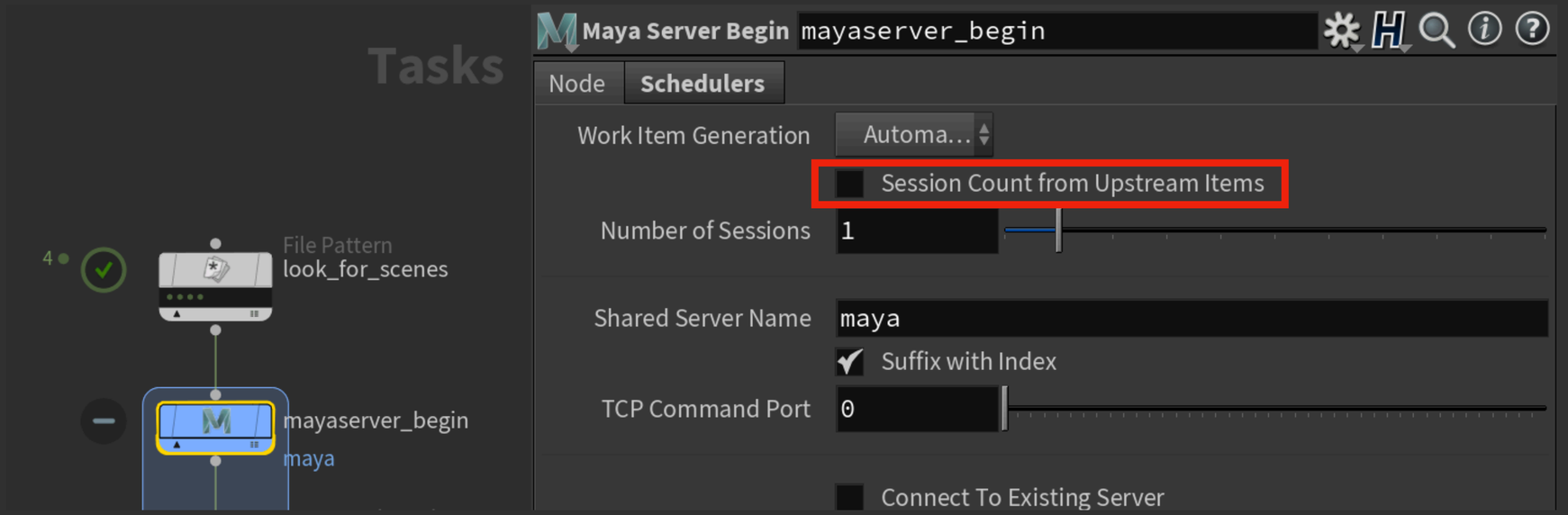


大事な設定：PDG_MAYAPY 環境変数



- Linux: /usr/autodesk/maya2019/bin/mayapy
- Mac: /Applications/Autodesk/maya2019/Maya.app/Contents/bin/mayapy
- Windows: C:/Program Files/Autodesk/Maya2019/bin/mayapy.exe

大事な設定：Session Count From Upstream Items



- **OFF**:　　各アイテムに別のMayaのプロセスを立ち上げる

  例：Maya起動→Open→作業→Save→Quit
  　　Maya起動→New→作業→Save→Quit
  　　Maya起動→Open→作業→Save→Quit
  　　} 同時にできる

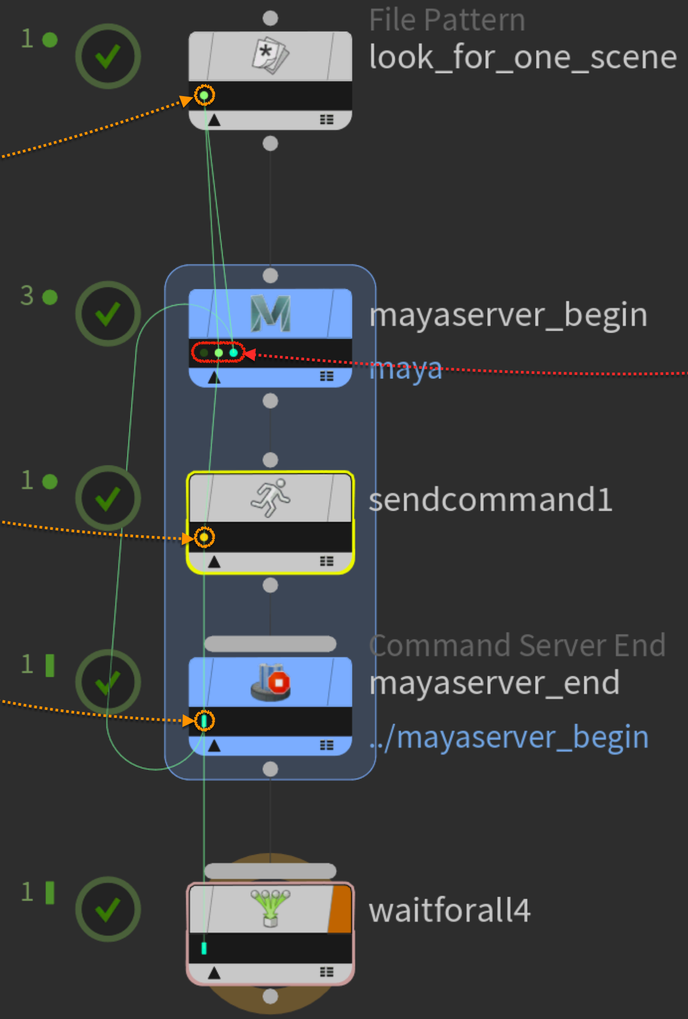
- **ON**:　　全てのアイテムを順番で一つのプロセスでやる

  例：Maya起動→Open→作業→Save→Open→作業→Save→New→作業　など

マニアックのコーナー：Server Items

一個づつ
（通常通り）

なんでここだけ三つ？？



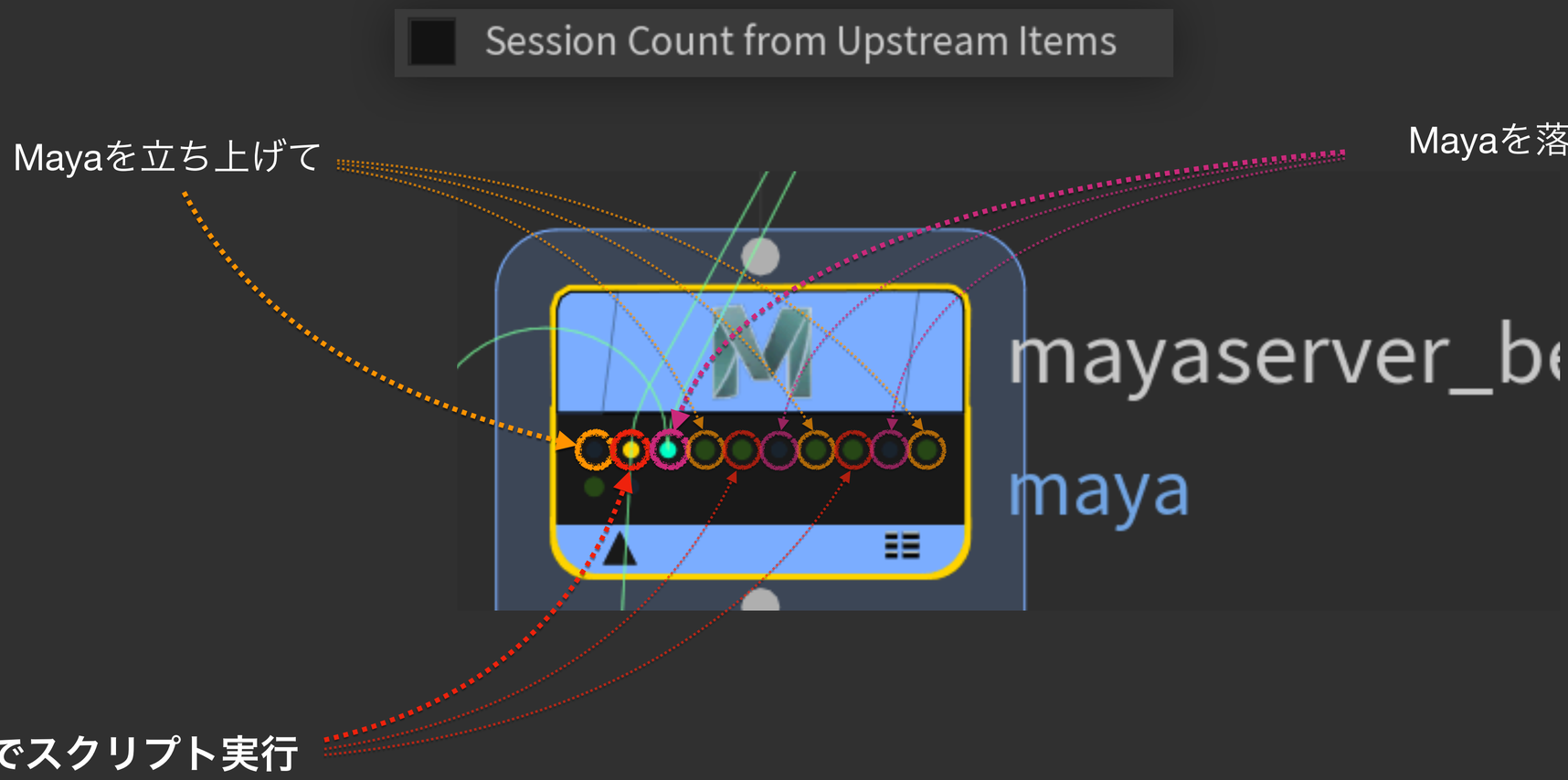
マニアックのコーナー：Server Items

End Serverアイテム
Mayaを落とす

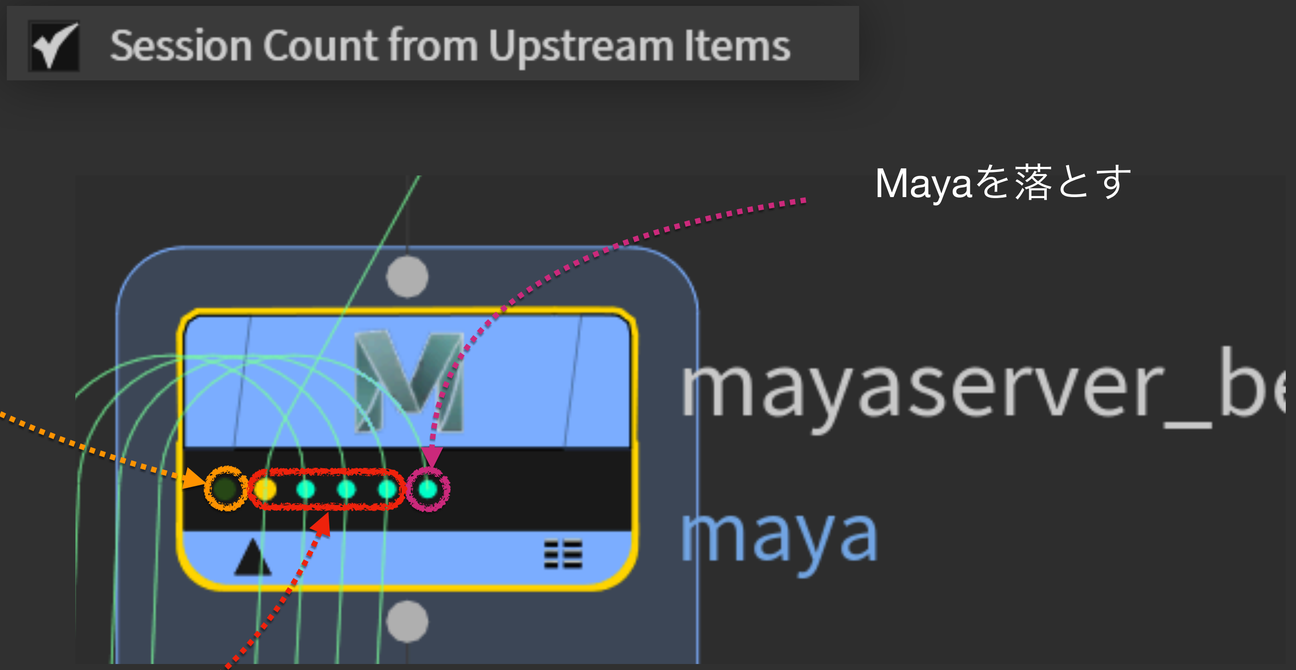
Begin Serverアイテム
ここでMayaが立ち上がる

Sessionアイテム
Command Sendのスクリプトの実行

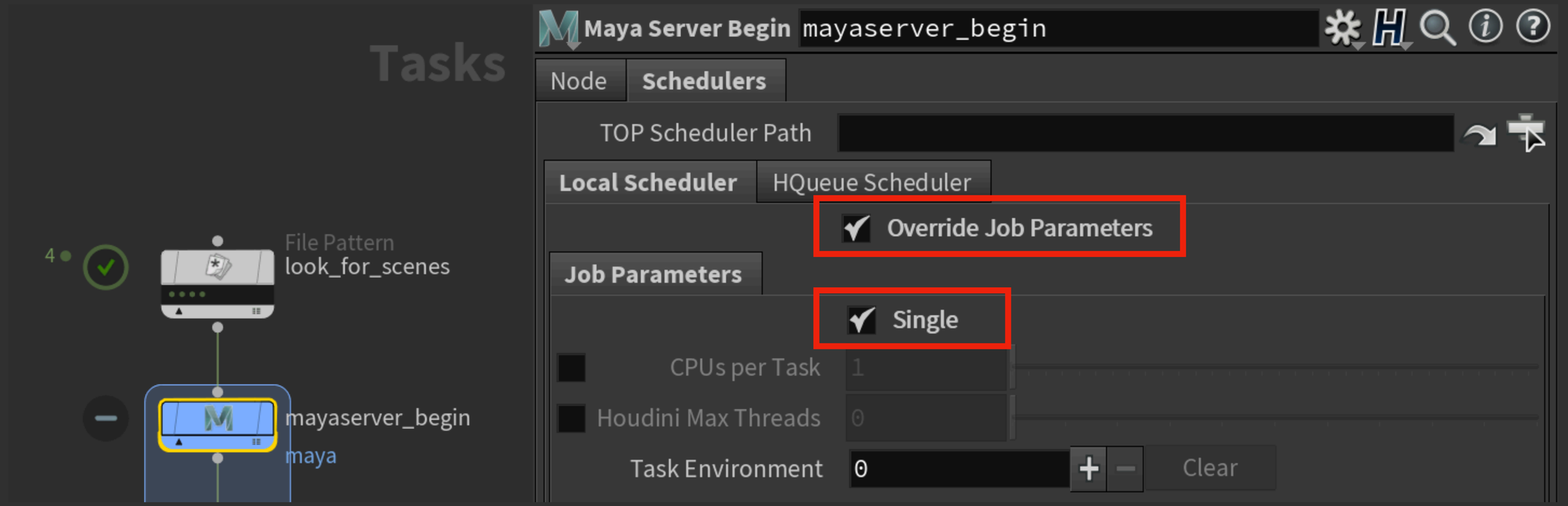
マニアックのコーナー：Server Items



マニアックのコーナー：Server Items

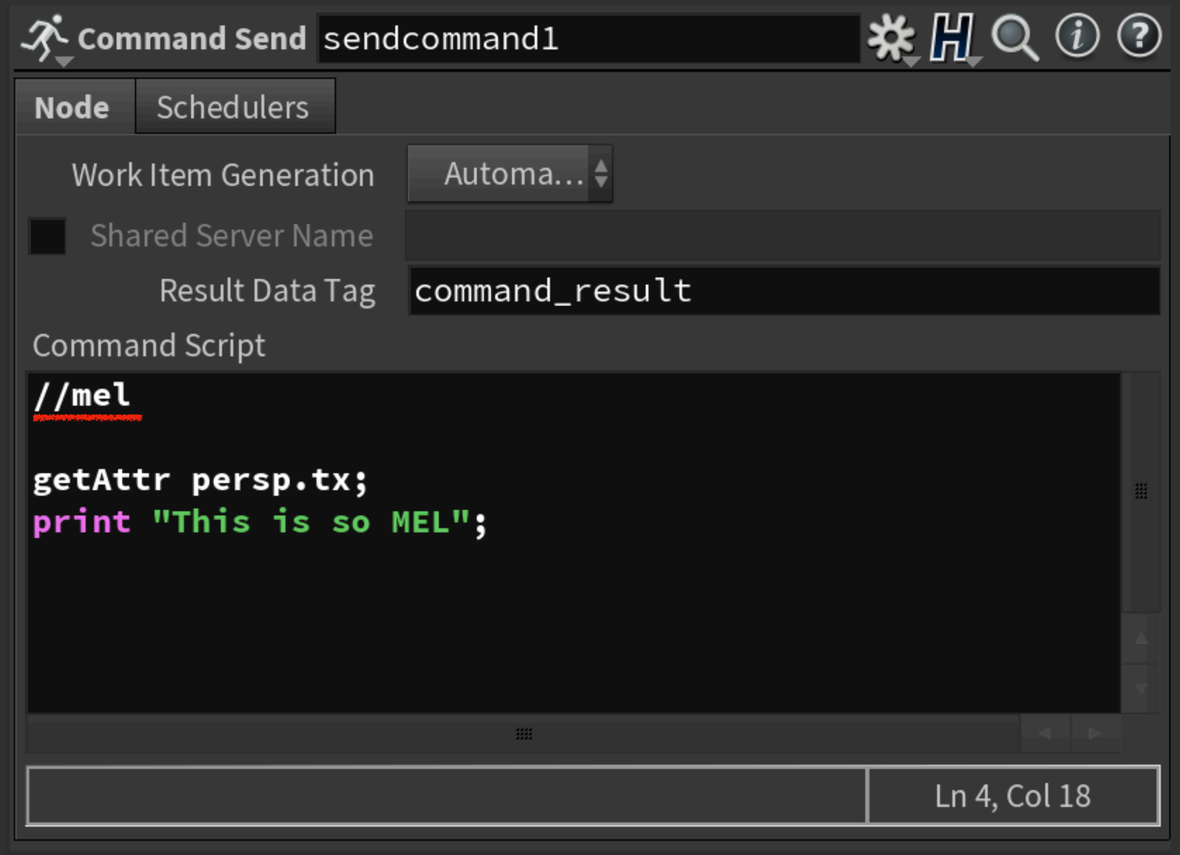


大事な設定：Local Scheduler Job Parameters



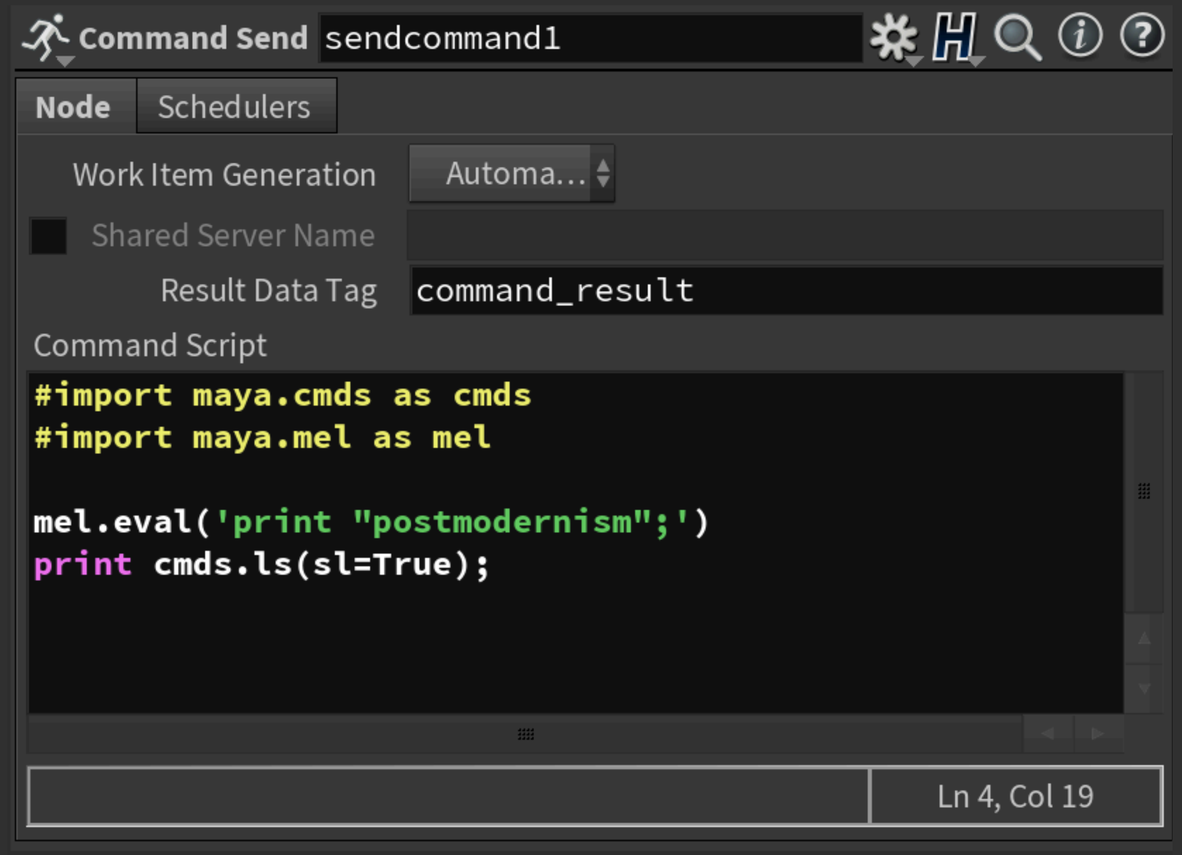
重いシーンを扱う場合、同時にMayaを一本しか立ち上がらないようにする設定

Command Send: MEL



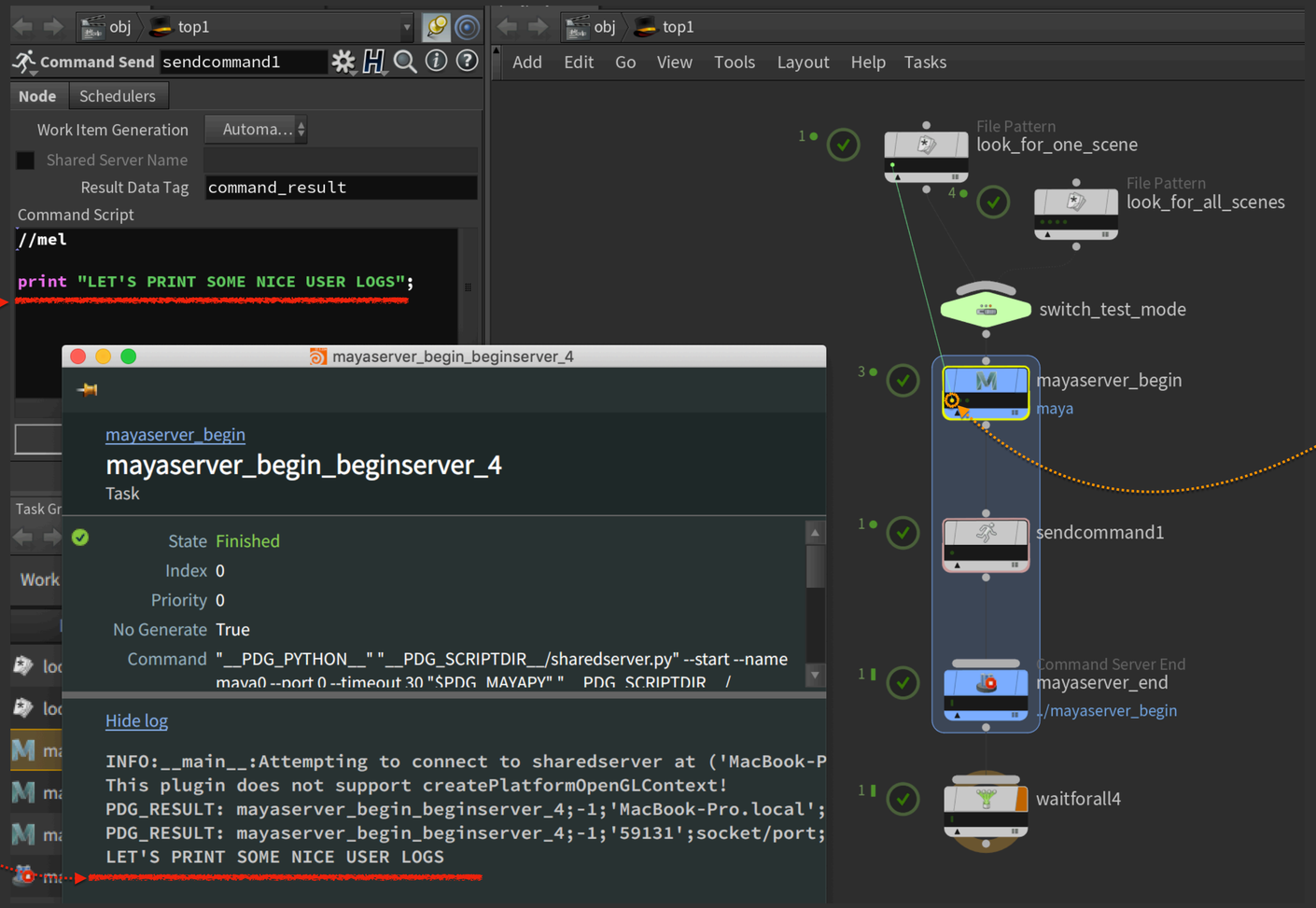
MELスクリプトを書く際、一番最初に **//mel** を書かなければならない

Command Send: Python



- //melを入れなければ、Pythonになる
- maya.cmdsとmaya.melは自動インポートされている

Command Send: Logs

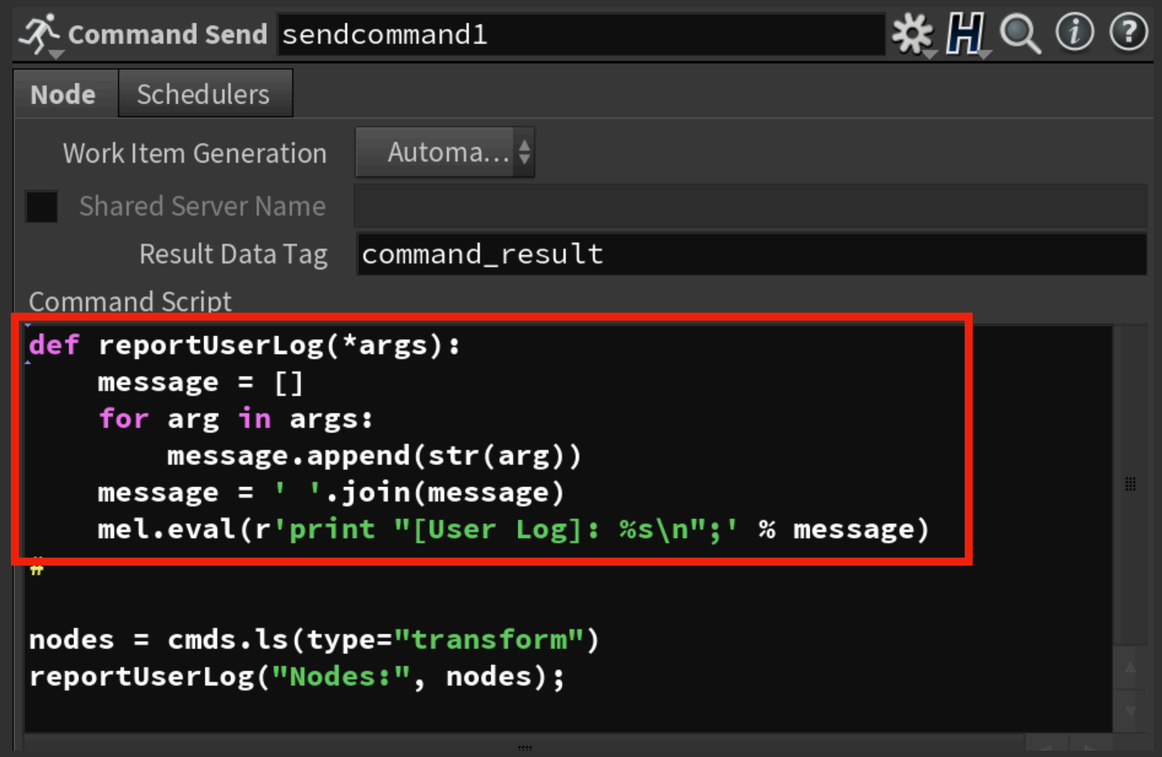


print出力

エラー以外のログは
Begin Serverのアイテムに入る

※私のPython知識不足か、Pythonのprintはアイテムのログに出てこない

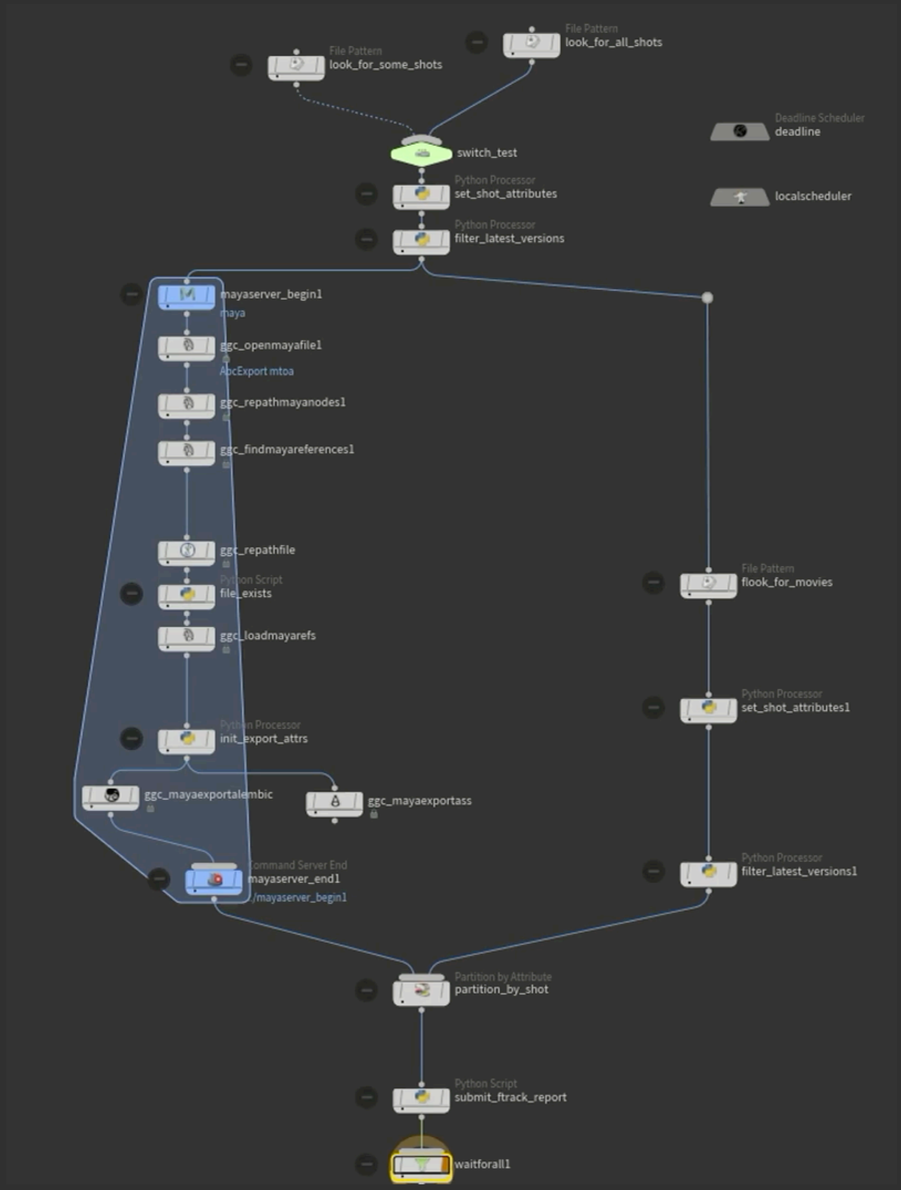
Command Send: python log



Hide log

```
INFO:__main__:Attempting to connect to sharedserver at ('MacBook-Pro.local', 59593)
This plugin does not support createPlatformOpenGLContext!
PDG_RESULT: mayaserver_begin_beginserver_6;-1;'MacBook-Pro.local';socket/ip;0
PDG_RESULT: mayaserver_begin_beginserver_6;-1;'59593';socket/port;0
[User Log]: Nodes: [u'front', u'persp', u'side', u'top']
```

プロダクションの例

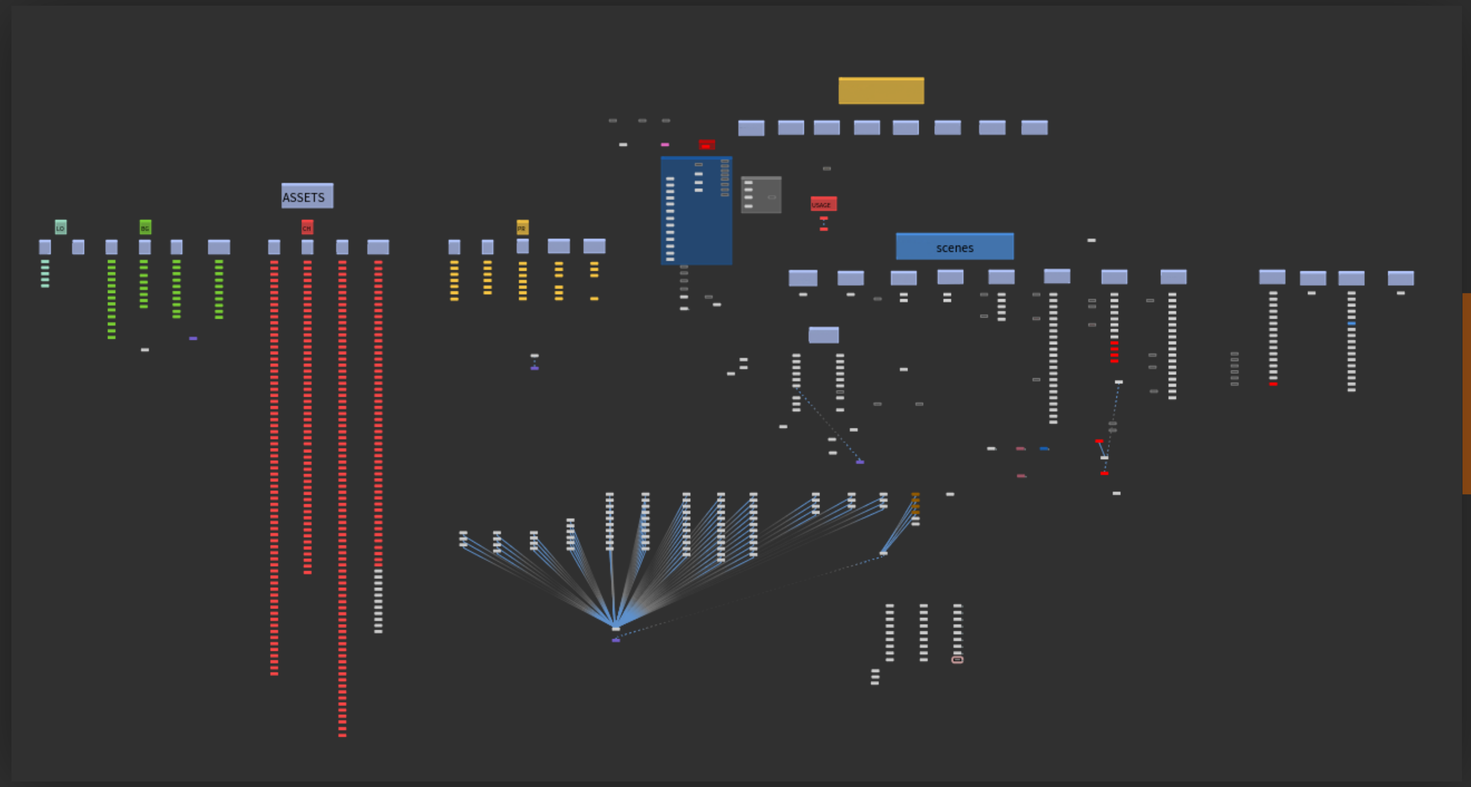

File Pattern
look_for_some_shots
File Pattern
look_for_all_shots
Deadline Scheduler
deadline
switch_test
Python Processor
set_shot_attributes
localscheduler
Python Processor
filter_latest_versions
mayaserver_begin1
maya
ggc_openmayafile1
AbcExport mtoa
ggc_repathmayanodes1
ggc_findmayareferences1
ggc_repathfile
Python Script
file_exists
ggc_loadmayarefs
File Pattern
flook_for_movies
Python Processor
set_shot_attributes1
Python Processor
init_export_attrs
ggc_mayaexportalembic
ggc_mayaexportass
Command Server End
mayaserver_end1
/mayaserver_begin1
Python Processor
filter_latest_versions1
Partition by Attribute
partition_by_shot
Python Script
submit_ftrack_report
waitforall1

TOPベースパイプラインのメリット

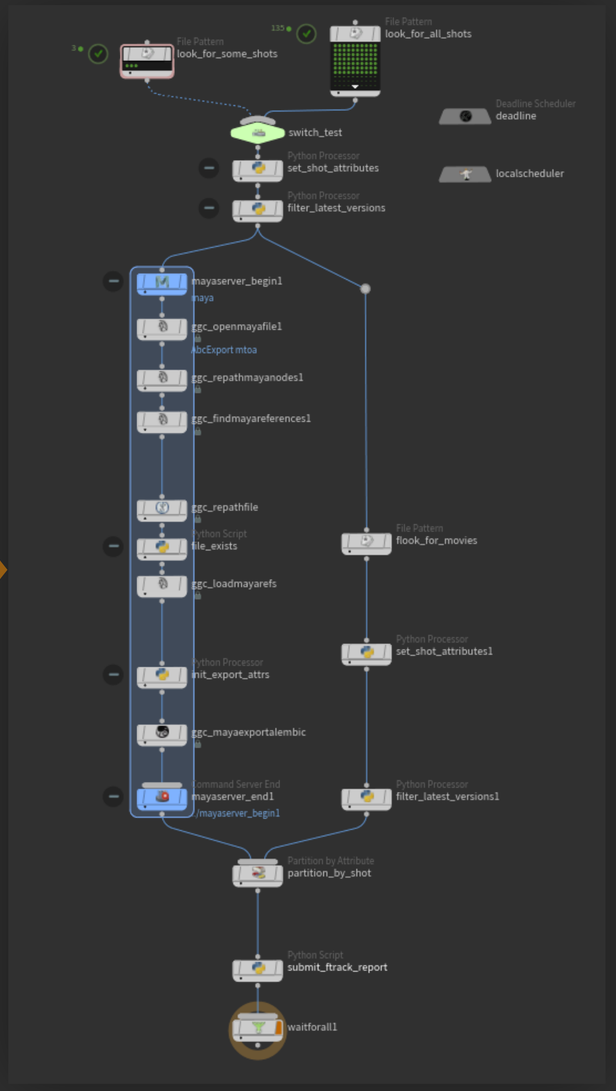
- パイプラインジャンキーじゃなくても組めるようになってきた
- パイプラインのタスクをプロシージャル的に扱える
- Pythonのバケモノスクリプトを書かなくていい
- SOHOは覚えなくていい
- 開発とデバッグに社内のリソースを前ほどかけなくていい
- 覚えたら他のことにも使える

結論の代わり

ROPs



TOPs



ご静聴ありがとうございました。

Q&A