This Jupyter Notebook contains the visualisations

Simulation models were created in simul8 and ROI distributions were done using AtRisk in Excel

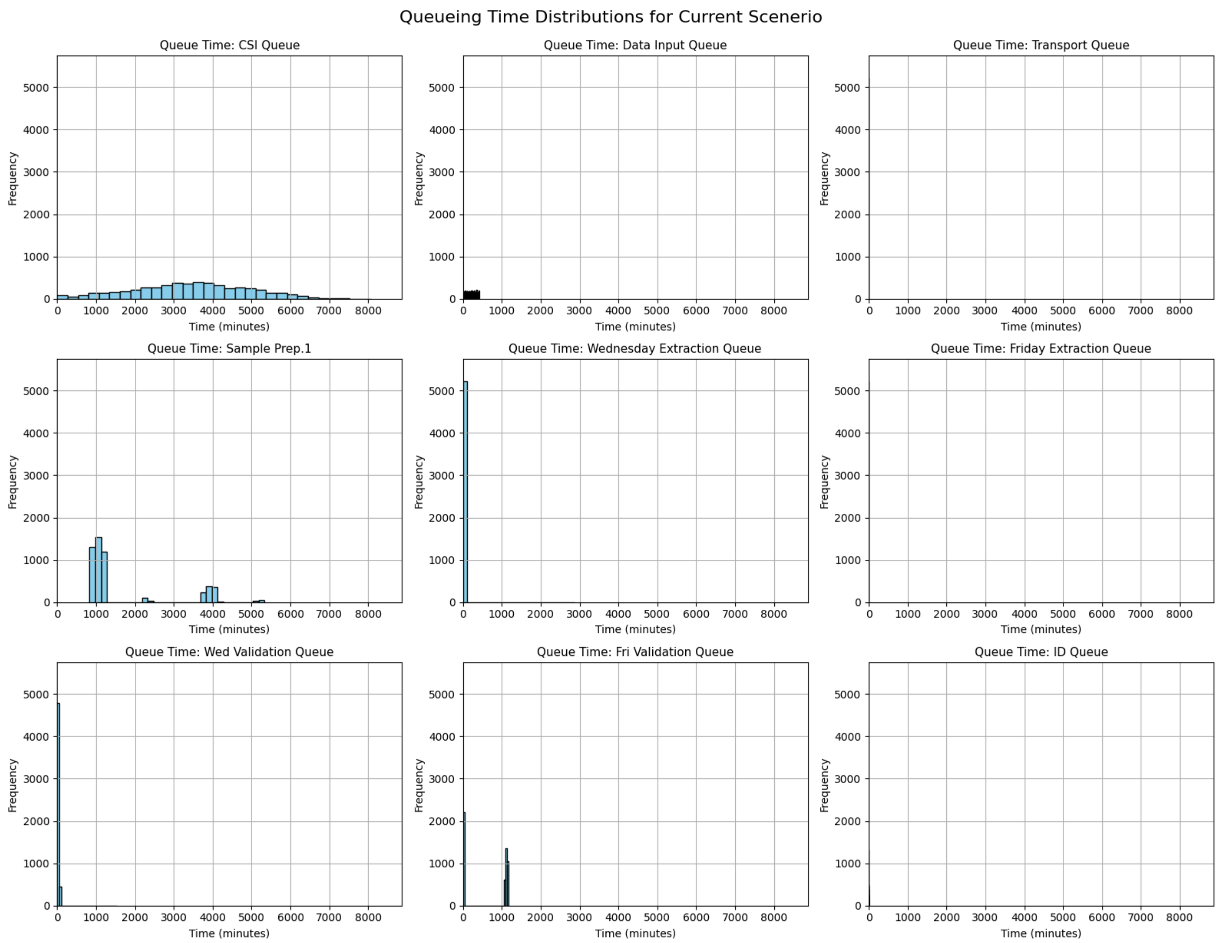
In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

In [3]:
```
df0 = pd.read_excel('C:/Users/123/Desktop/OA/DataSets/Current_Scenerio.xlsx')
queue_cols0 = ['CSI Queue', 'Data Input Queue', 'Transport Queue', 'Sample Prep.1',
avg_queues0 = df0[queue_cols0].mean()


df1 = pd.read_excel('C:/Users/123/Desktop/OA/DataSets/s1.xlsx')
queue_cols1 = ['CSI_Sample_Collection_Queue', 'Data_Input_Queue', 'Transport_Queue'
avg_queues1 = df1[queue_cols1].mean()


df2 = pd.read_excel('C:/Users/123/Desktop/OA/DataSets/S2.xlsx')
queue_cols2 = ['CSI_Sample_Collection_Queue_Time', 'Courier_Collection_Queue_Time',
avg_queues2 = df2[queue_cols2].mean()


df3 = pd.read_excel('C:/Users/123/Desktop/OA/DataSets/S3.xlsx')
queue_cols3 = ['Time_in_CSI_Queue', 'Time_in_Courier_Queue', 'Time_in_Sample_Prep_Q
avg_queues3 = df3[queue_cols3].mean()


df4 = pd.read_excel('C:/Users/123/Desktop/OA/DataSets/S4.xlsx')
queue_cols4 = ['CSI_Queue_Time', 'Courier_Queue_Time', 'Sample_Prep_Queue_Time', 'R
avg_queues4 = df4[queue_cols4].mean()


df5 = pd.read_excel('C:/Users/123/Desktop/OA/DataSets/S5.xlsx')
queue_cols5 = ['CSI_Queue_Time', 'Courier_Queue_Time', 'Sample_Prep_Queue_Time', 'R
avg_queues5 = df5[queue_cols5].mean()
```

Current Scenerio

In [9]:
```
all_data = pd.concat([df0[col] for col in queue_cols0])
xlim = (0, all_data.max().max() * 1.1)  # Add 10% margin
ylim = (0, 1.1 * max(df0[col].value_counts().max() for col in queue_cols0))

fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(16, 12))
axs = axs.flatten()
```

```
for i, col in enumerate(queue_cols0):
    axs[i].hist(df0[col].dropna(), bins=30, edgecolor='black', color='skyblue')
    axs[i].set_title(f'Queue Time: {col}', fontsize=11)
    axs[i].set_xlim(xlim)
    axs[i].set_ylim(ylim)
    axs[i].set_xlabel('Time (minutes)')
    axs[i].set_ylabel('Frequency')
    axs[i].grid(True)

# Hide any unused subplots (e.g. if only 8 plots)
for j in range(len(queue_cols0), len(axs)):
    fig.delaxes(axs[j])

plt.tight_layout()
plt.suptitle('Queueing Time Distributions for Current Scenerio', fontsize=16, y=1.0
plt.show()
```

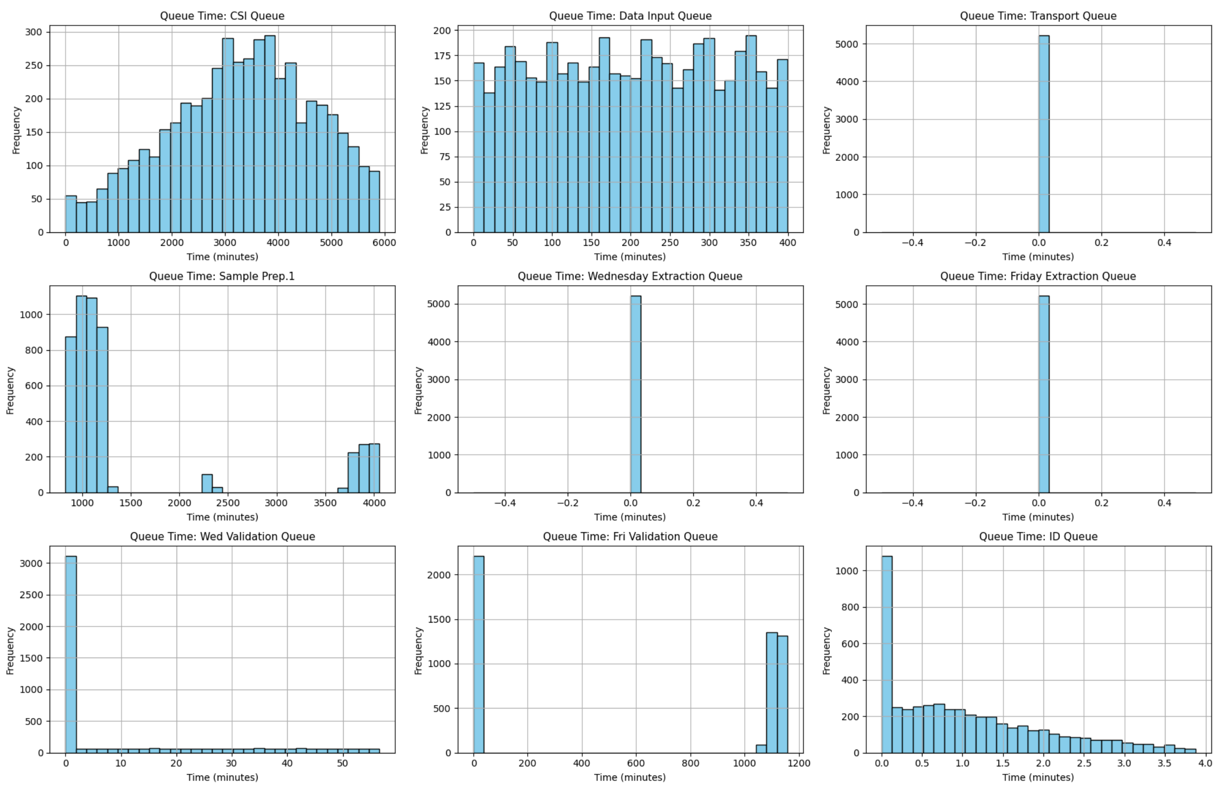


In [10]:

```
fig, axs = plt.subplots(3, 3, figsize=(18, 12))
axs = axs.flatten()

for i, col in enumerate(queue_cols0):
    data = df0[col].dropna()

    # Clip to 95th percentile to avoid huge x-axis spread
    max_clip = data.quantile(0.95)
    data_clipped = data[data <= max_clip]
```

```
    axs[i].hist(data_clipped, bins=30, edgecolor='black', color='skyblue')
    axs[i].set_title(f'Queue Time: {col}', fontsize=11)
    axs[i].set_xlabel('Time (minutes)')
    axs[i].set_ylabel('Frequency')
    axs[i].grid(True)

# Hide unused subplots
for j in range(len(queue_cols0), len(axs)):
    fig.delaxes(axs[j])

plt.suptitle('Queueing Time Distributions for Current Scenerio(Scaled))', fontsize=
plt.tight_layout()
plt.show()
```

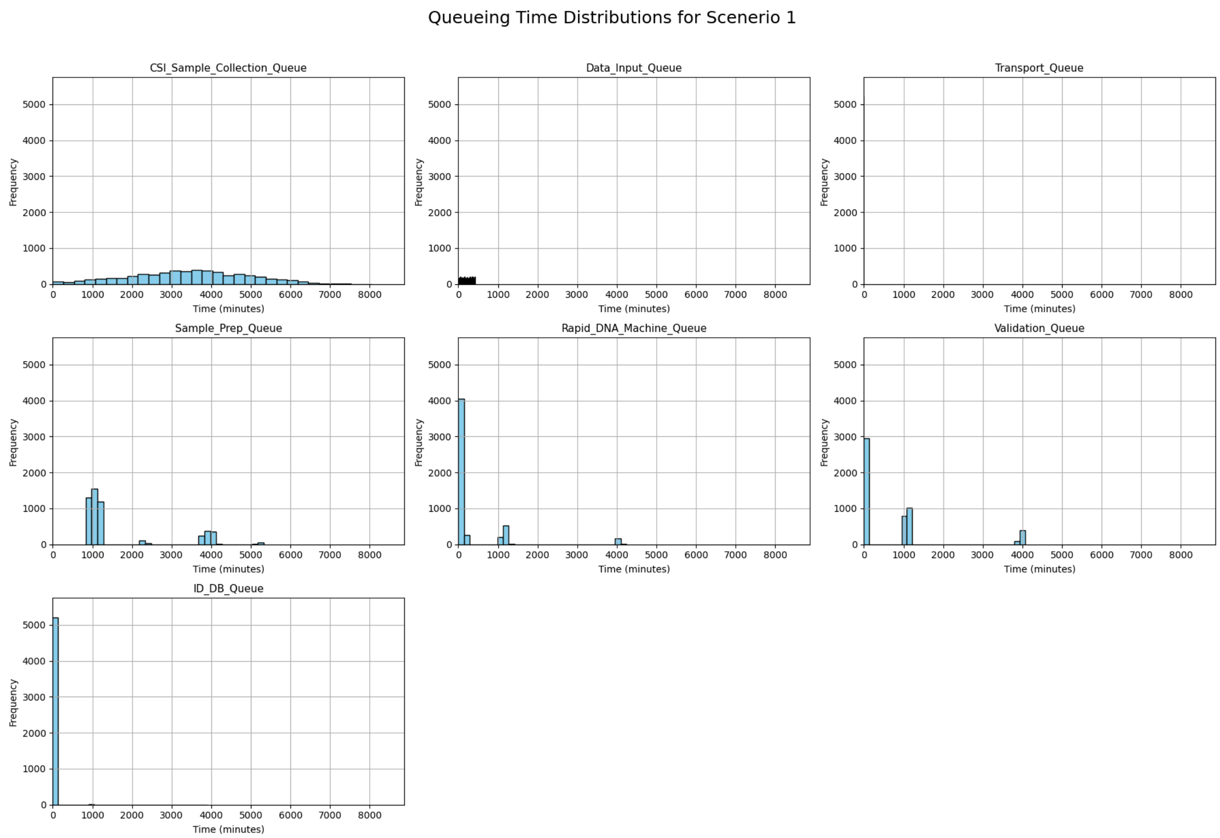
Queueing Time Distributions for Current Scenerio(Scaled))



Scenerio 1

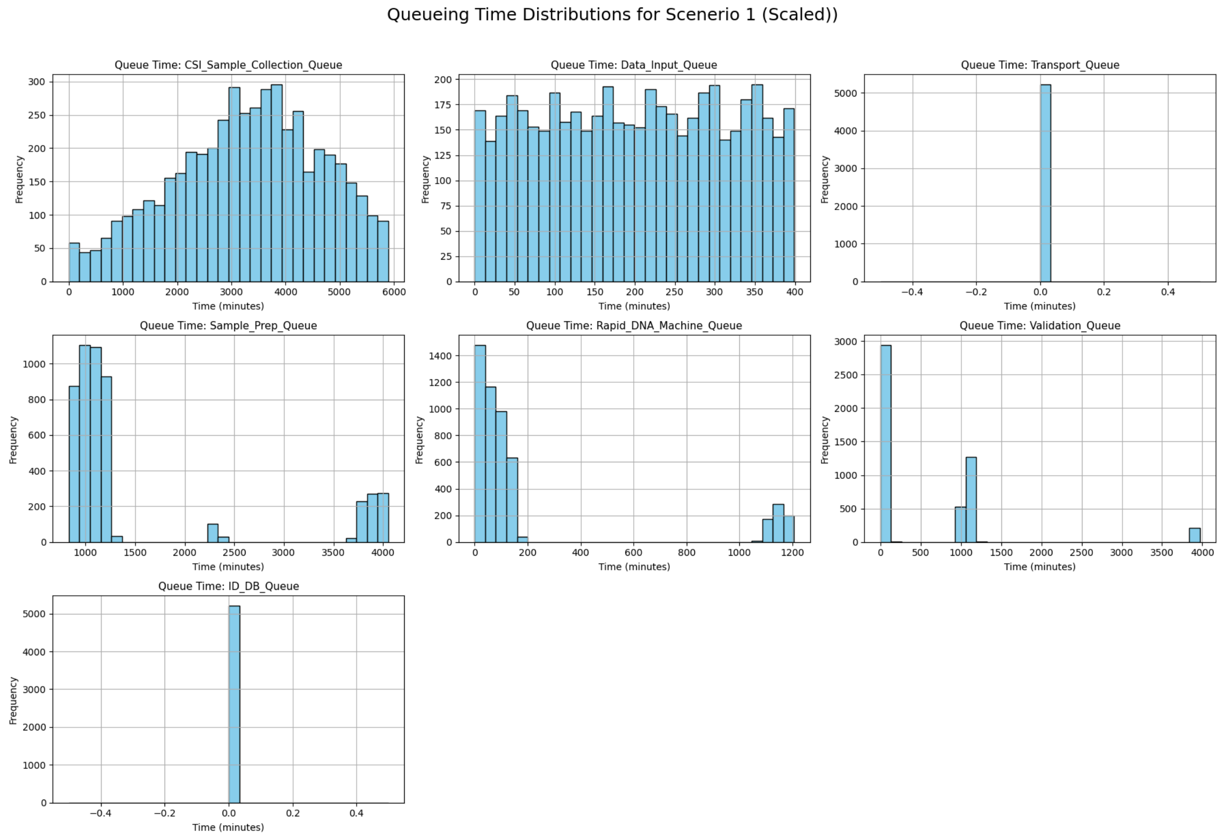
In [12]:
```
all_values = pd.concat([df1[col].dropna() for col in queue_cols1])
xlim = (0, all_values.max() * 1.1)
ylim = (0, max(df1[col].value_counts().max() for col in queue_cols1) * 1.1)


fig, axs = plt.subplots(3, 3, figsize=(18, 12))  # 3x3 for 9 stages
axs = axs.flatten()

for i, col in enumerate(queue_cols1):
    axs[i].hist(df1[col].dropna(), bins=30, edgecolor='black', color='skyblue')
    axs[i].set_title(f'{col}', fontsize=11)
    axs[i].set_xlim(xlim)
    axs[i].set_ylim(ylim)
```

```
    axs[i].set_xlabel('Time (minutes)')
    axs[i].set_ylabel('Frequency')
    axs[i].grid(True)

# Hide any extra subplot boxes (if fewer than 9 columns)
for j in range(len(queue_cols1), len(axs)):
    fig.delaxes(axs[j])

plt.suptitle('Queueing Time Distributions for Scenerio 1', fontsize=18, y=1.02)
plt.tight_layout()
plt.show()
```



In [13]:
```
fig, axs = plt.subplots(3, 3, figsize=(18, 12))
axs = axs.flatten()

for i, col in enumerate(queue_cols1):
    data = df1[col].dropna()

    # Clip to 95th percentile to avoid huge x-axis spread
    max_clip = data.quantile(0.95)
    data_clipped = data[data <= max_clip]

    axs[i].hist(data_clipped, bins=30, edgecolor='black', color='skyblue')
    axs[i].set_title(f'Queue Time: {col}', fontsize=11)
    axs[i].set_xlabel('Time (minutes)')
    axs[i].set_ylabel('Frequency')
    axs[i].grid(True)

# Hide unused subplots
for j in range(len(queue_cols1), len(axs)):
```

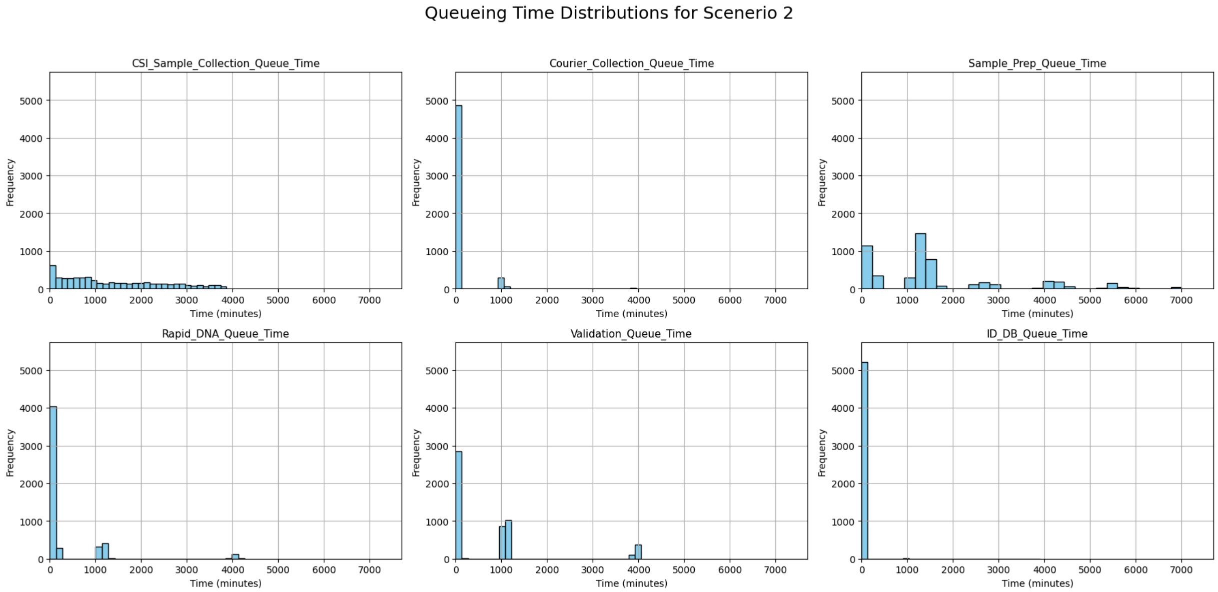
```
    fig.delaxes(axs[j])

plt.suptitle('Queueing Time Distributions for Scenerio 1 (Scaled))', fontsize=18, y
plt.tight_layout()
plt.show()
```



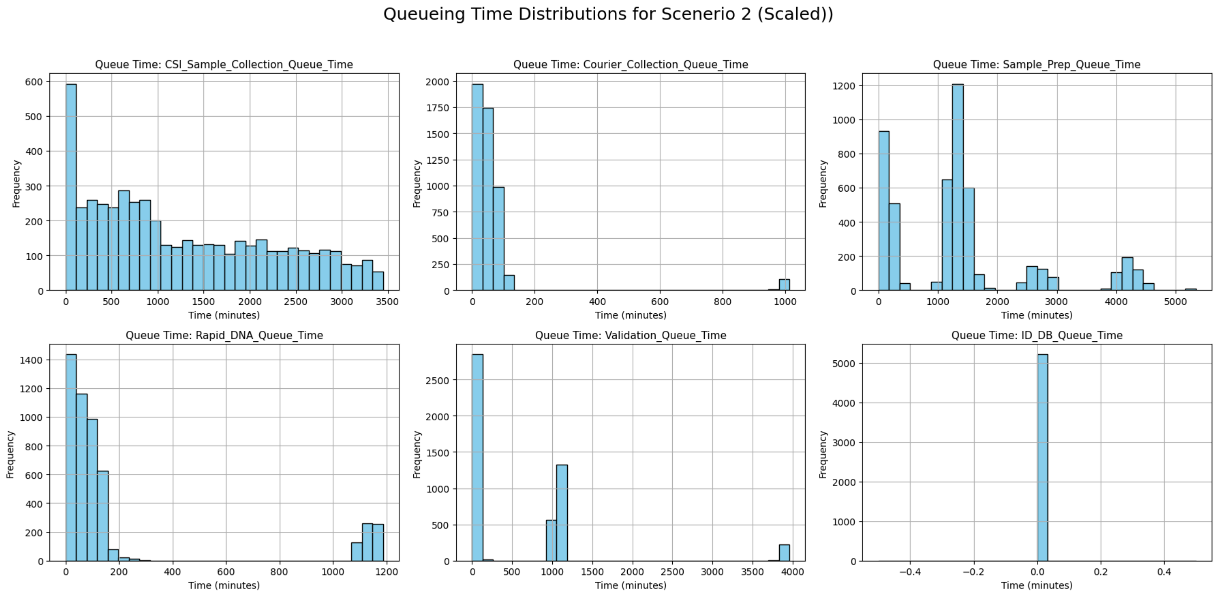
Scenerio 2

In [15]:
```
all_values = pd.concat([df2[col].dropna() for col in queue_cols2])
xlim = (0, all_values.max() * 1.1)
ylim = (0, max(df2[col].value_counts().max() for col in queue_cols2) * 1.1)

fig, axs = plt.subplots(3, 3, figsize=(18, 12))  # 3x3 for 9 stages
axs = axs.flatten()

for i, col in enumerate(queue_cols2):
    axs[i].hist(df2[col].dropna(), bins=30, edgecolor='black', color='skyblue')
    axs[i].set_title(f'{col}', fontsize=11)
    axs[i].set_xlim(xlim)
    axs[i].set_ylim(ylim)
    axs[i].set_xlabel('Time (minutes)')
    axs[i].set_ylabel('Frequency')
    axs[i].grid(True)

# Hide any extra subplot boxes (if fewer than 9 columns)
for j in range(len(queue_cols2), len(axs)):
    fig.delaxes(axs[j])

plt.suptitle('Queueing Time Distributions for Scenerio 2', fontsize=18, y=1.02)
```

```
plt.tight_layout()
plt.show()
```

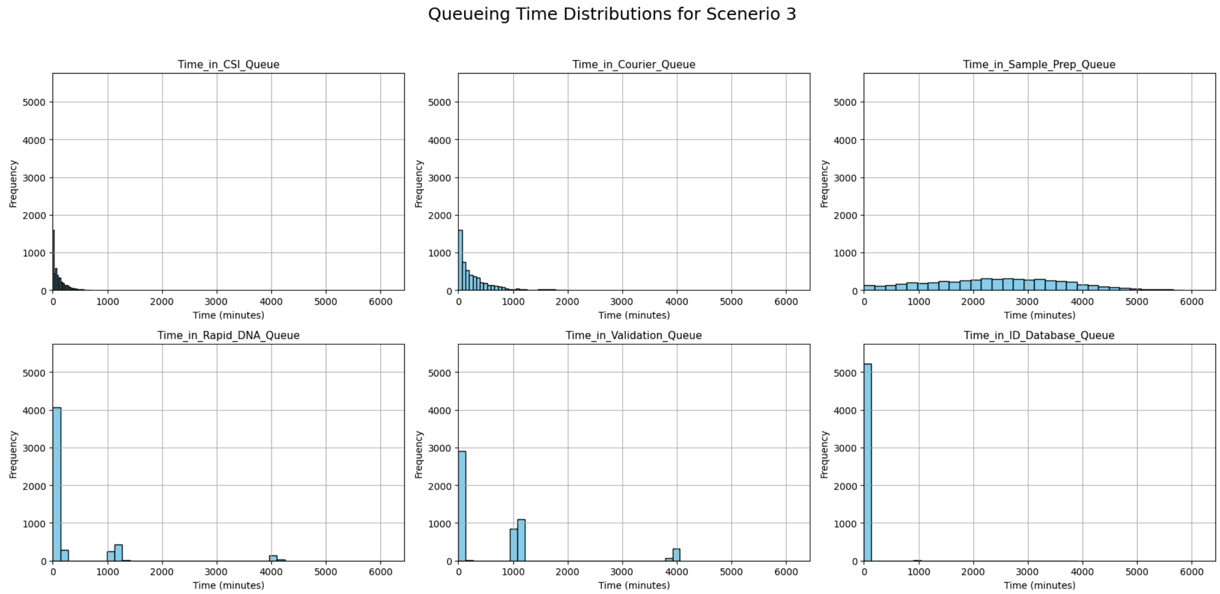


In [16]:
```
fig, axs = plt.subplots(3, 3, figsize=(18, 12))
axs = axs.flatten()

for i, col in enumerate(queue_cols2):
    data = df2[col].dropna()

    # Clip to 95th percentile to avoid huge x-axis spread
    max_clip = data.quantile(0.95)
    data_clipped = data[data <= max_clip]

    axs[i].hist(data_clipped, bins=30, edgecolor='black', color='skyblue')
    axs[i].set_title(f'Queue Time: {col}', fontsize=11)
    axs[i].set_xlabel('Time (minutes)')
    axs[i].set_ylabel('Frequency')
    axs[i].grid(True)

# Hide unused subplots
for j in range(len(queue_cols2), len(axs)):
    fig.delaxes(axs[j])

plt.suptitle('Queueing Time Distributions for Scenerio 2 (Scaled))', fontsize=18, y
plt.tight_layout()
plt.show()
```



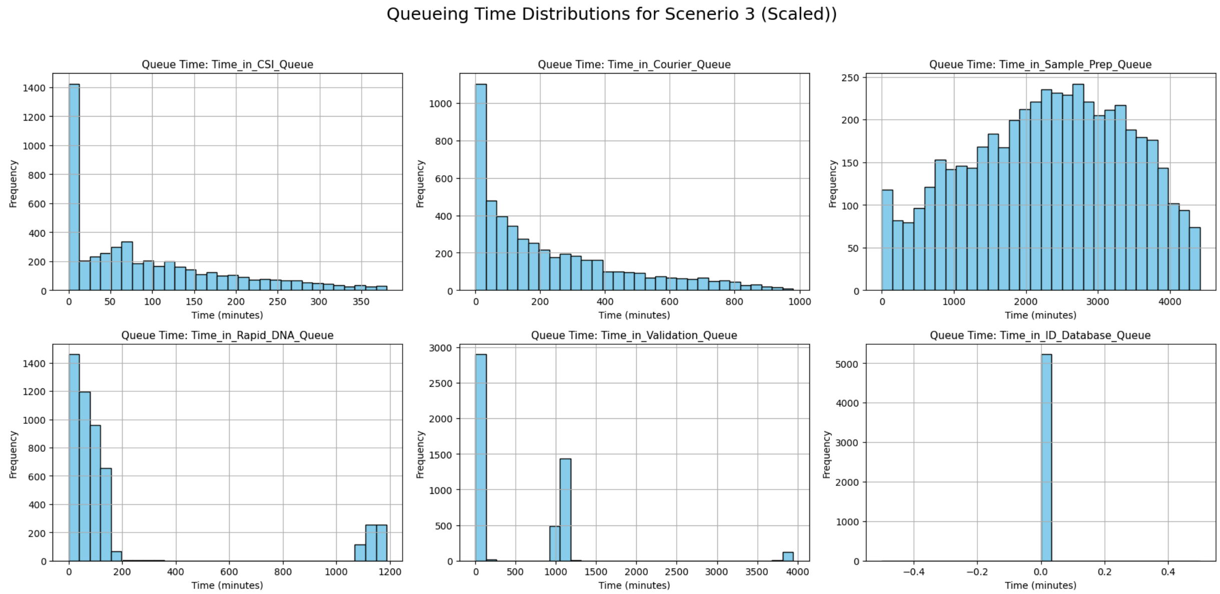
Scenerio 3

In [17]:
```
all_values = pd.concat([df3[col].dropna() for col in queue_cols3])
xlim = (0, all_values.max() * 1.1)
ylim = (0, max(df3[col].value_counts().max() for col in queue_cols3) * 1.1)

fig, axs = plt.subplots(3, 3, figsize=(18, 12))  # 3x3 for 9 stages
axs = axs.flatten()

for i, col in enumerate(queue_cols3):
    axs[i].hist(df3[col].dropna(), bins=30, edgecolor='black', color='skyblue')
    axs[i].set_title(f'{col}', fontsize=11)
    axs[i].set_xlim(xlim)
    axs[i].set_ylim(ylim)
    axs[i].set_xlabel('Time (minutes)')
    axs[i].set_ylabel('Frequency')
    axs[i].grid(True)

# Hide any extra subplot boxes (if fewer than 9 columns)
for j in range(len(queue_cols3), len(axs)):
    fig.delaxes(axs[j])

plt.suptitle('Queueing Time Distributions for Scenerio 3', fontsize=18, y=1.02)
plt.tight_layout()
plt.show()
```



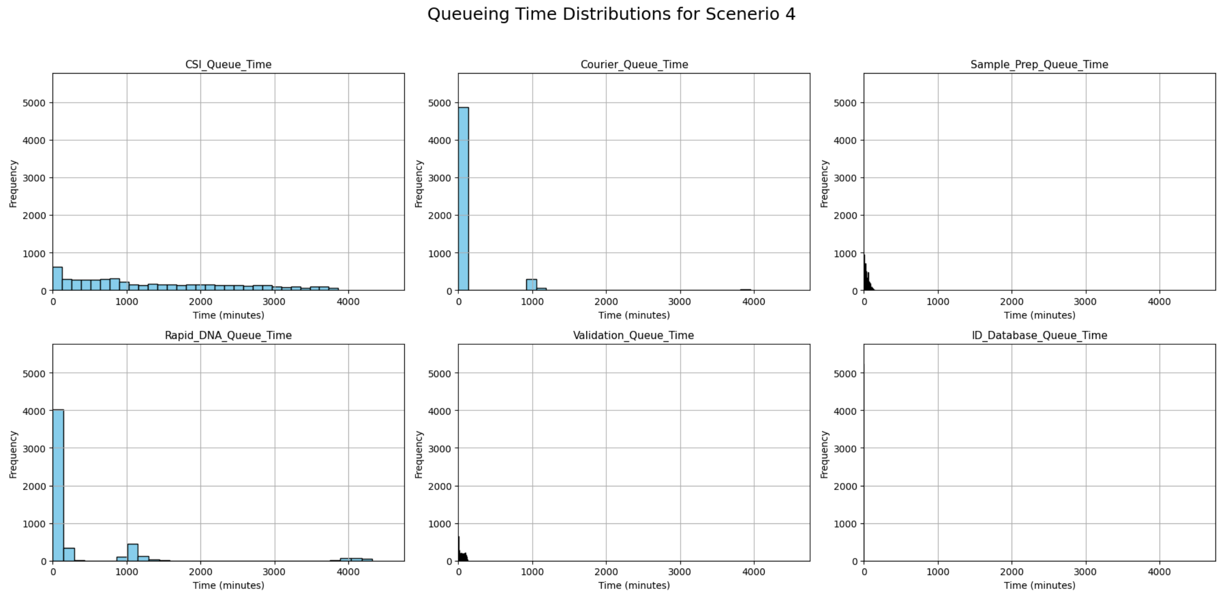
In [18]:

```
fig, axs = plt.subplots(3, 3, figsize=(18, 12))
axs = axs.flatten()

for i, col in enumerate(queue_cols3):
    data = df3[col].dropna()

    # Clip to 95th percentile to avoid huge x-axis spread
    max_clip = data.quantile(0.95)
    data_clipped = data[data <= max_clip]

    axs[i].hist(data_clipped, bins=30, edgecolor='black', color='skyblue')
    axs[i].set_title(f'Queue Time: {col}', fontsize=11)
    axs[i].set_xlabel('Time (minutes)')
    axs[i].set_ylabel('Frequency')
    axs[i].grid(True)

# Hide unused subplots
for j in range(len(queue_cols3), len(axs)):
    fig.delaxes(axs[j])

plt.suptitle('Queueing Time Distributions for Scenerio 3 (Scaled))', fontsize=18, y
plt.tight_layout()
plt.show()
```



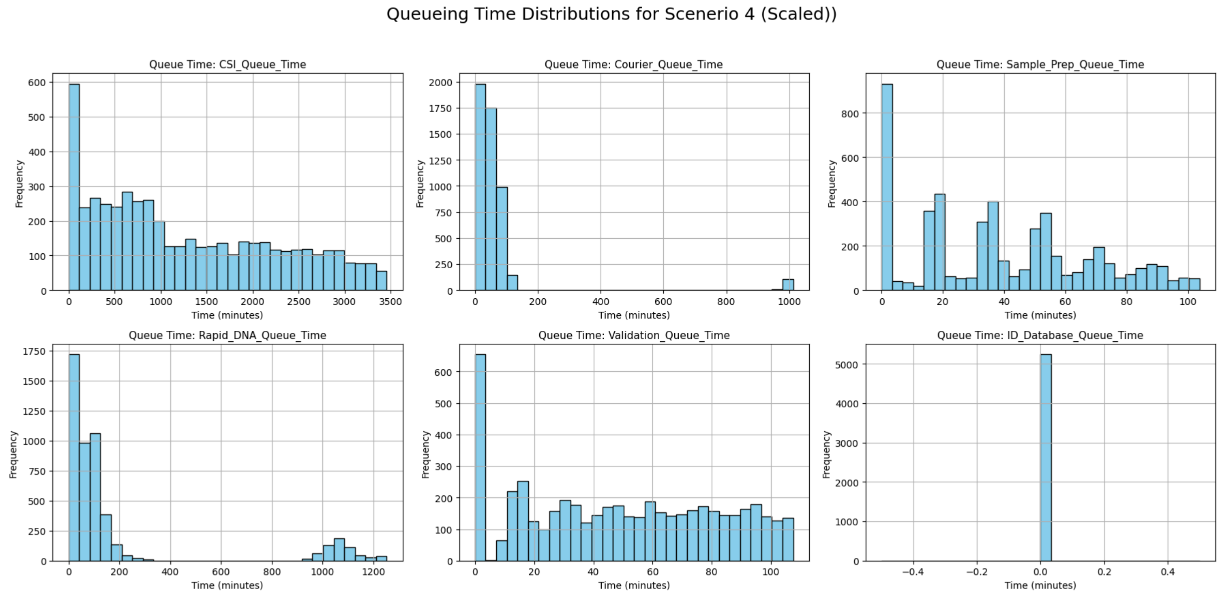
Scenerio 4

In [19]:
```
all_values = pd.concat([df4[col].dropna() for col in queue_cols4])
xlim = (0, all_values.max() * 1.1)
ylim = (0, max(df4[col].value_counts().max() for col in queue_cols4) * 1.1)

fig, axs = plt.subplots(3, 3, figsize=(18, 12))  # 3x3 for 9 stages
axs = axs.flatten()

for i, col in enumerate(queue_cols4):
    axs[i].hist(df4[col].dropna(), bins=30, edgecolor='black', color='skyblue')
    axs[i].set_title(f'{col}', fontsize=11)
    axs[i].set_xlim(xlim)
    axs[i].set_ylim(ylim)
    axs[i].set_xlabel('Time (minutes)')
    axs[i].set_ylabel('Frequency')
    axs[i].grid(True)

# Hide any extra subplot boxes (if fewer than 9 columns)
for j in range(len(queue_cols4), len(axs)):
    fig.delaxes(axs[j])

plt.suptitle('Queueing Time Distributions for Scenerio 4', fontsize=18, y=1.02)
plt.tight_layout()
plt.show()
```



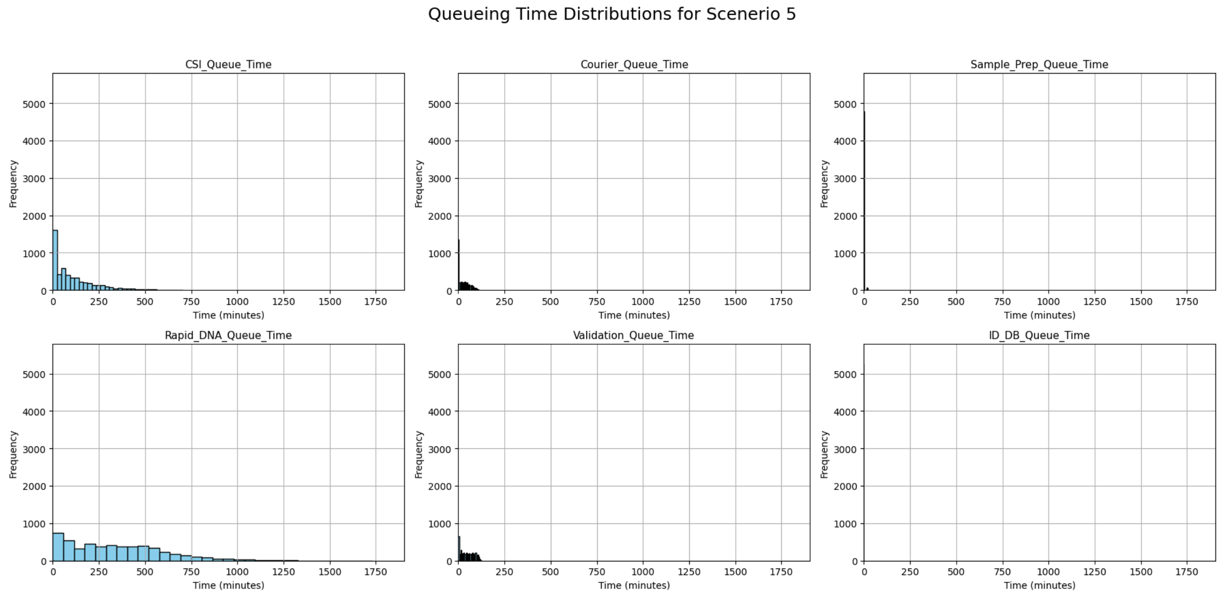
In [20]:

```
fig, axs = plt.subplots(3, 3, figsize=(18, 12))
axs = axs.flatten()

for i, col in enumerate(queue_cols4):
    data = df4[col].dropna()

    # Clip to 95th percentile to avoid huge x-axis spread
    max_clip = data.quantile(0.95)
    data_clipped = data[data <= max_clip]

    axs[i].hist(data_clipped, bins=30, edgecolor='black', color='skyblue')
    axs[i].set_title(f'Queue Time: {col}', fontsize=11)
    axs[i].set_xlabel('Time (minutes)')
    axs[i].set_ylabel('Frequency')
    axs[i].grid(True)

# Hide unused subplots
for j in range(len(queue_cols4), len(axs)):
    fig.delaxes(axs[j])

plt.suptitle('Queueing Time Distributions for Scenerio 4 (Scaled))', fontsize=18, y
plt.tight_layout()
plt.show()
```



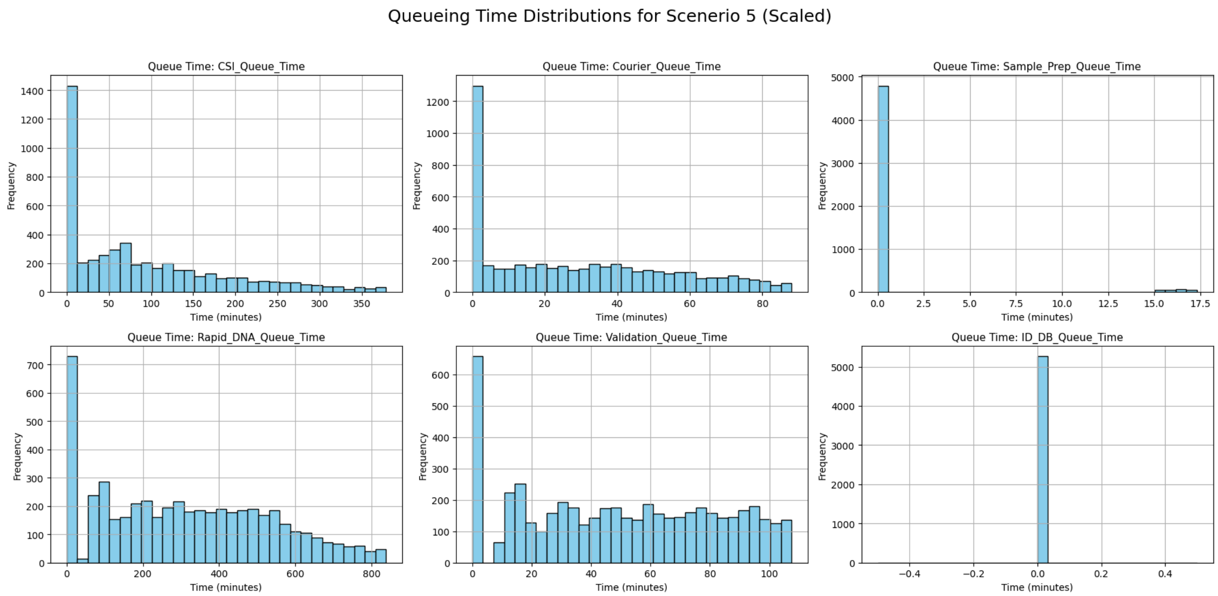
Scenerio 5

In [21]:
```
all_values = pd.concat([df5[col].dropna() for col in queue_cols5])
xlim = (0, all_values.max() * 1.1)
ylim = (0, max(df5[col].value_counts().max() for col in queue_cols5) * 1.1)

fig, axs = plt.subplots(3, 3, figsize=(18, 12))  # 3x3 for 9 stages
axs = axs.flatten()

for i, col in enumerate(queue_cols5):
    axs[i].hist(df5[col].dropna(), bins=30, edgecolor='black', color='skyblue')
    axs[i].set_title(f'{col}', fontsize=11)
    axs[i].set_xlim(xlim)
    axs[i].set_ylim(ylim)
    axs[i].set_xlabel('Time (minutes)')
    axs[i].set_ylabel('Frequency')
    axs[i].grid(True)

# Hide any extra subplot boxes (if fewer than 9 columns)
for j in range(len(queue_cols5), len(axs)):
    fig.delaxes(axs[j])

plt.suptitle('Queueing Time Distributions for Scenerio 5', fontsize=18, y=1.02)
plt.tight_layout()
plt.show()
```



In [22]:

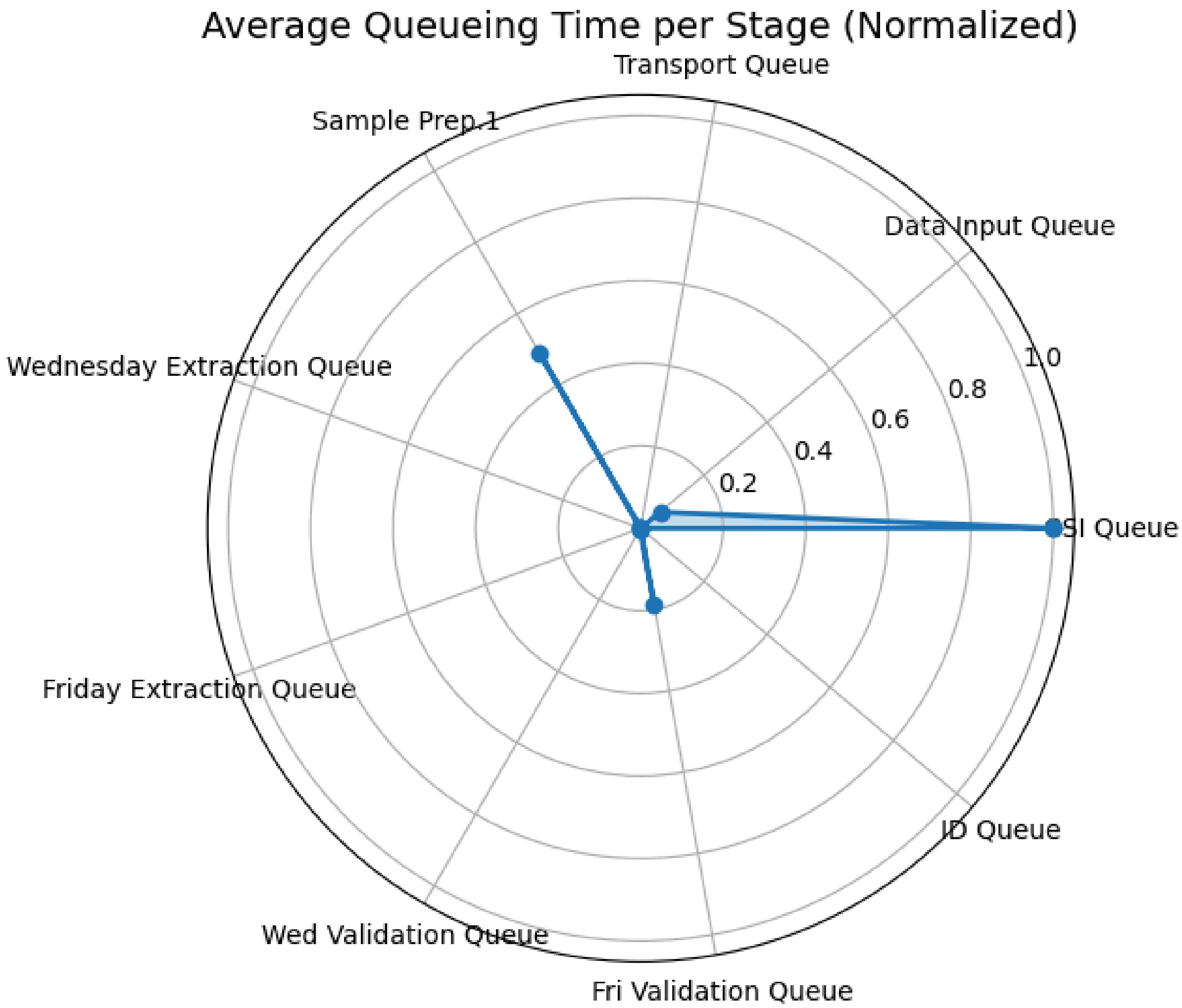
```
fig, axs = plt.subplots(3, 3, figsize=(18, 12))
axs = axs.flatten()

for i, col in enumerate(queue_cols5):
    data = df5[col].dropna()

    # Clip to 95th percentile to avoid huge x-axis spread
    max_clip = data.quantile(0.95)
    data_clipped = data[data <= max_clip]

    axs[i].hist(data_clipped, bins=30, edgecolor='black', color='skyblue')
    axs[i].set_title(f'Queue Time: {col}', fontsize=11)
    axs[i].set_xlabel('Time (minutes)')
    axs[i].set_ylabel('Frequency')
    axs[i].grid(True)

# Hide unused subplots
for j in range(len(queue_cols5), len(axs)):
    fig.delaxes(axs[j])

plt.suptitle('Queueing Time Distributions for Scenerio 5 (Scaled)', fontsize=18, y=
plt.tight_layout()
plt.show()
```



////////////////////////////////////

Queue times spidergram for current scenerio

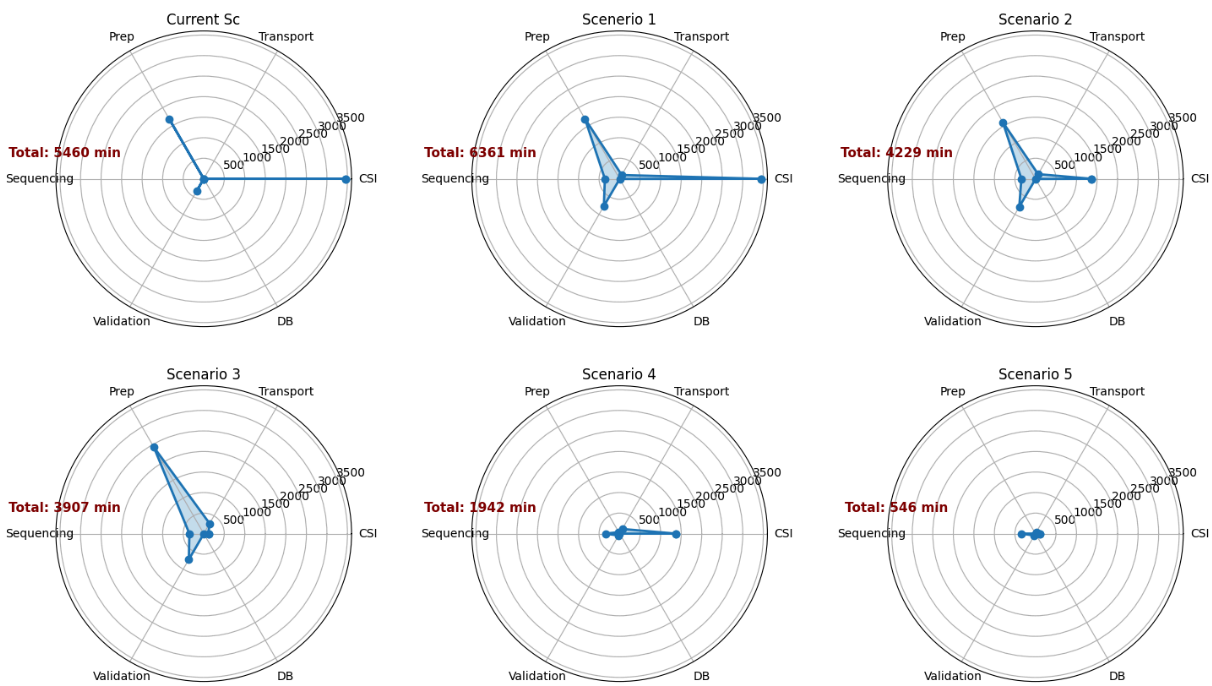
```
In [25]:
queue_cols = ['CSI Queue', 'Data Input Queue', 'Transport Queue', 'Sample Prep.1',
avg_queues = df0[queue_cols0].mean()
normalized = avg_queues / avg_queues.max()

labels = normalized.index.tolist()
values = normalized.values.tolist()

# Close the loop
values += values[:1]
angles = np.linspace(0, 2 * np.pi, len(labels), endpoint=False).tolist()
angles += angles[:1]

# Plot
fig, ax = plt.subplots(figsize=(6, 6), subplot_kw=dict(polar=True))
ax.plot(angles, values, 'o-', linewidth=2)
ax.fill(angles, values, alpha=0.25)
ax.set_thetagrids(np.degrees(angles[:-1]), labels)
ax.set_title('Average Queueing Time per Stage (Normalized)', size=14)
plt.show()
```



Comparative Spidergram for Queue Times

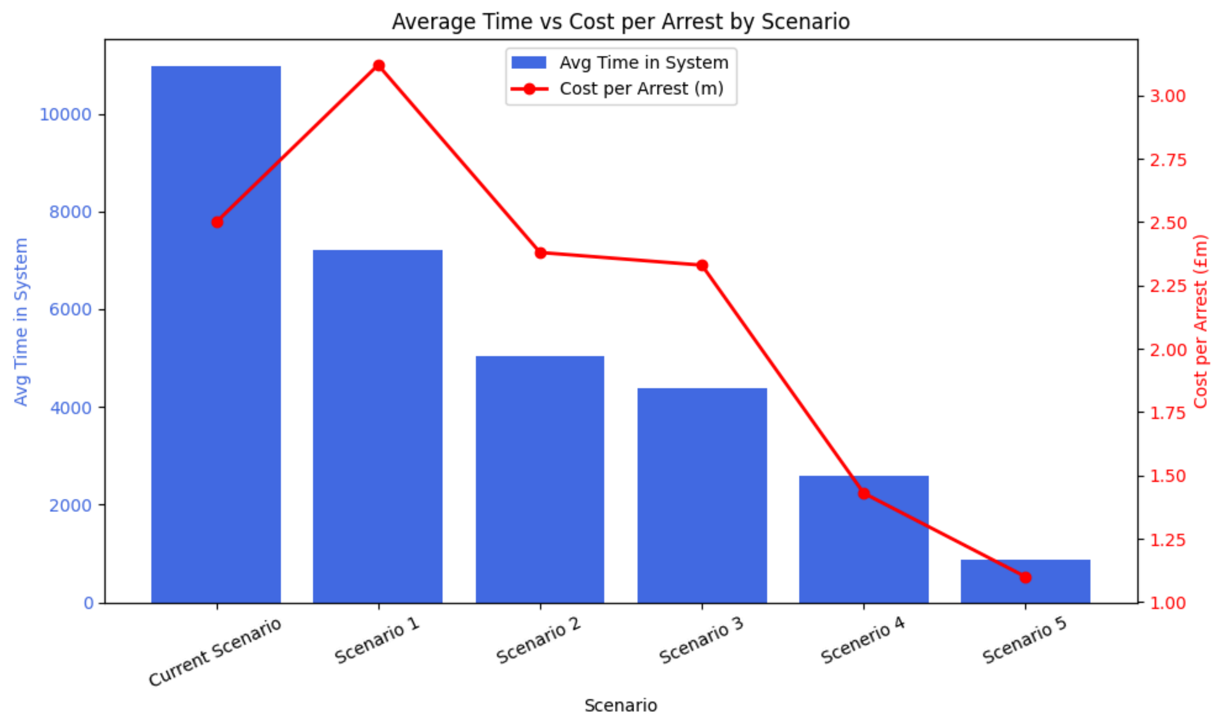
In [27]:
```
labels = ['CSI', 'Transport', 'Prep', 'Sequencing', 'Validation', 'DB']

# Ensure all scenario values have EXACTLY 6 values
spider_data = {
    'Current Sc': [
        avg_queues0.loc['CSI Queue'].item(),
        avg_queues0.loc['Transport Queue'].item(),
        avg_queues0.loc['Sample Prep.1'].item(),
        avg_queues0.loc[['Wednesday Extraction Queue', 'Friday Extraction Queue']].
        avg_queues0.loc[['Wed Validation Queue', 'Fri Validation Queue']].mean().it
        avg_queues0.loc['ID Queue'].item()
    ],
    'Scenerio 1':  [
        avg_queues1.loc['CSI_Sample_Collection_Queue'].item(),
        avg_queues1.loc[['Data_Input_Queue', 'Transport_Queue']].mean().item(),
        avg_queues1.loc['Sample_Prep_Queue'].item(),
        avg_queues1.loc['Rapid_DNA_Machine_Queue'].item(),
        avg_queues1.loc[['Validation_Queue']].mean().item(),
        avg_queues1.loc['ID_DB_Queue'].item()
    ],
    'Scenario 2': avg_queues2.values.tolist()[:6],
```

```
    'Scenario 3': avg_queues3.values.tolist()[:6],
    'Scenario 4': avg_queues4.values.tolist()[:6],
    'Scenario 5': avg_queues5.values.tolist()[:6]
}

# Build consistent angles
angles = np.linspace(0, 2 * np.pi, len(labels), endpoint=False).tolist()
angles += angles[:1]

# Plotting
fig, axs = plt.subplots(2, 3, subplot_kw=dict(polar=True), figsize=(18, 10))
angles = np.linspace(0, 2 * np.pi, len(labels), endpoint=False).tolist()
angles += angles[:1]

for ax, (scenario, values) in zip(axs.flat, spider_data.items()):
    values_plot = values + values[:1]
    ax.plot(angles, values_plot, 'o-', linewidth=2)
    ax.fill(angles, values_plot, alpha=0.25)
    ax.set_thetagrids(np.degrees(angles[:-1]), labels)
    ax.set_title(scenario)
    ax.set_ylim(0, 3600)
    total_time = sum(values)
    ax.text(3, ax.get_ylim()[1] * 0.95, f"Total: {int(total_time)} min",
            ha='center', va='bottom', fontsize=11, fontweight='bold', color='maroon
```



In [16]:
```
data = {
    'Scenario': ['Current Scenario', 'Scenario 1', 'Scenario 2', 'Scenario 3', 'Sce
    'Avg_Time_in_System': [10971, 7211, 5027,4372, 2584, 871],
    'Cost_per_Arrest_m': [2.5, 3.12, 2.38, 2.33, 1.43, 1.1]
}

df = pd.DataFrame(data)

# Create figure and axis
fig, ax1 = plt.subplots(figsize=(10, 6))
```

```
# Bar plot for average time
bar = ax1.bar(df['Scenario'], df['Avg_Time_in_System'], color='royalblue', label='A
ax1.set_xlabel('Scenario')
ax1.set_ylabel('Avg Time in System', color='royalblue')
ax1.tick_params(axis='y', labelcolor='royalblue')

# Rotate x labels if needed
plt.xticks(rotation=25)

# Secondary axis for cost per arrest
ax2 = ax1.twinx()
line = ax2.plot(df['Scenario'], df['Cost_per_Arrest_m'], color='red', marker='o', l
ax2.set_ylabel('Cost per Arrest (£m)', color='red')
ax2.tick_params(axis='y', labelcolor='red')

# Combine legends
lines, labels = ax1.get_legend_handles_labels()
lines2, labels2 = ax2.get_legend_handles_labels()
ax1.legend(lines + lines2, labels + labels2, loc='upper center')

plt.title('Average Time vs Cost per Arrest by Scenario')
plt.tight_layout()
plt.show()
```




```
In [ ]:
```